Albert-Ludwigs-Universität Freiburg

Programmierumgebung für den Informatikunterricht an Gymnasien

Wissenschaftliche Arbeit

von Yannik Ries

bei Prof. Dr. Hannah Bast

10. Oktober 2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig angefertigt und nur die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken, gegebenenfalls auch elektronischen Medien, entnommen sind, sind von mir durch Angabe der Quelle und des Zugriffsdatums sowie dem Ausdruck der ersten Seite belegt; sie liegen zudem für den Zeitraum von 2 Jahren entweder auf einem elektronischen Speichermedium im PDF-Format oder in gedruckter Form vor.

Yannik Ries

Abstract

An Gymnasien in Baden-Württemberg wird der Informatikunterricht stark ausgebaut. Für den Unterricht werden neue, zeitgemäße und didaktisch sinnvolle Werkzeuge benötigt. Diese Arbeit befasst sich daher mit der Erstellung einer Programmierumgebung für den Unterricht. Zu diesem Zweck wurden zunächst Anforderungen an eine Programmierumgebung im Umfeld der Schule analysiert und bereits existierende Programmierumgebungen auf diese Anforderungen geprüft. Unter Beachtung der ermittelten Anforderungen und mit Kenntnis der wertvollen Komponenten der Alternativen wurde eine Programmierumgebung mithilfe des PyQt Frameworks erstellt. Sie erlaubt den Schülerinnen und Schülern (SuS) die praktische Auseinandersetzung mit den informationstechnischen Inhalten des Lehrplans mit Python. Sie bietet dabei zahlreiche, das Verständnis fördernde Features wie schrittweise Ausführung des Codes, schülergerechte Fehlermeldungen und Visualisierung des Schülercodes.
Abschließend wurde die Programmierumgebung mit einer Schülergruppe im Unterricht getestet und evaluiert.

1 Einleitung

1.1 Motivation

Seit dem Schuljahr 2018/2019 kann an baden-württembergischen Gymnasien neben dem sprachlichen und dem naturwissenschaftlichen Profil auch das neue Profilfach IMP (Informatik Mathematik Physik) angeboten werden. Diese Entwicklung ändert den Umgang mit der Informatik grundlegend und erhöht zusammen mit dem Aufbaukurs Informatik in der siebten Klassenstufe den Bedarf an Unterrichtsmaterial enorm. Da von den SuS auch aktiv programmiert werden soll (vgl. Kapitel 2.1) schließt dieser Bedarf Programmierumgebungen ein. Klassischerweise wird im Unterricht Java als Programmiersprache verwendet. Es existieren zahlreiche Tools und Programmierumgebungen, die die SuS beim Schreiben von Java Code unterstützen können. In Kapitel 3 wird im Detail auf solche bereits vorhandenen Programmierumgebungen und deren Vor- und Nachteile eingegangen. Da Python eine der syntaktisch einfachsten und am schnellsten wachsenden Sprachen ist, bietet sie sich vermeintlich für den Unterricht an. Python war bereits 2014 an US Universitäten die beliebteste Wahl für Einstiegskurse [24]. Laut dem Popularity of Programming Language Index hat Python mittlerweile sogar Java als die Programmiersprache, deren Tutorials am häufigsten auf Google gesucht werden, überholt [26]. Auch bei anderen Rankings steigt Python in den letzten Jahren stetig im Rang [25]. Aufgrund dieser Entwicklung ist der Mangel einer für den Unterricht geeigneten Python-Programmierumgebung zu beheben.

1.2 Ziel

Das Ziel dieser Arbeit ist es, eine speziell auf die Unterrichtsthemen der IMP zugeschnittene Programmierumgebung für Python zu erstellen. Somit soll Python im Unterricht als eine valide Alternative zu dem bisher hauptsächlich verwendeten Java etabliert werden. Die genauen Anforderungen an eine solche Programmierumgebung werden in Kapitel 2 genauer untersucht. Zusammenfassend lässt sich die Programmierumgebung als Werkzeug beschreiben, das:

- die praktische Umsetzung der in der IMP vermittelten Informatik Konzepte wie Algorithmen auf Graphen, sortieren von Listen, Schleifen und Funktionen ermöglicht.
- für die erfolgreiche Nutzung die Vermittlung theoretischen Wissens dieser Konzepte in vorangehenden Schulstunden voraussetzt.
- im Unterricht von den SuS selbst bedient wird.
- den SuS konkrete Programmieraufgaben und Hilfestellung zu diesen Aufgaben zur Verfügung stellt.

- keine spezielle, standardmäßig nicht in Python enthaltene Syntax voraussetzt.
- den von SuS geschriebenen Code ausführt.
- den SuS bei dem Verständnis unterstützt durch:
  - Visualisierung des Codes.
  - Verlangsamte Ausführung des Codes.
  - Deutliche Fehlermeldungen mit Hilfestellung.
- von Lehrkräften an die Anforderungen des eigenen Unterrichts angepasst werden kann.
- zukünftig im Leistungsumfang ergänzt werden kann.

Die Herausforderungen bei der Erreichung dieses Ziels liegen nicht bei Laufzeitoptimierungen oder Umgang mit großen Datenmengen, sondern vielmehr bei der Unvorhersehbarkeit des Verhaltens der Schüler und Schülerinnen (SuS), der Vielfältigkeit der Aufgaben, und der Gestaltung einer geeigneten Lernumgebung.

1.3 Aufbau der Arbeit

In Kapitel 2 werden sämtliche Anforderungen an die Programmierumgebung zusammengefasst, die von SuS, Lehrkräften, der Bildungspsychologie und dem Staat gestellt werden. Anschließend werden in Kapitel 3 bereits existierende Programmierumgebungen für Lernende aufgeführt und mit den Anforderungen verglichen. Dabei werden besonders gelungene Aspekte hervorgehoben. Kapitel 4 stellt die im Rahmen dieser Arbeit erstellte Programmierumgebung vor. Dabei wird zunächst der Leistungsumfang präsentiert und im Anschluss der Aufbau sowie der Entwicklungs- / Entscheidungsprozess dargelegt. Kapitel 5 analysiert die Programmierumgebung auf ihre Anwendbarkeit bei den gegebenen Rahmenbedingungen und evaluiert sie aufgrund einer Benutzerstudie. Abschließend bietet Kapitel 6 einen Ausblick auf die mögliche zukünftige Entwicklung der Programmierumgebung.

2 Anforderungen und Rahmenbedingungen

2.1 Schulpolitischer Hintergrund

Bis vor wenigen Jahren wurde Informatik außer an speziellen Schulen entweder nicht oder nur als zweistündiges Fach in der Oberstufe unterrichtet. Da bei einem solchen Fach keine zentralen Prüfungen stattfinden, konnte die Lehrkraft frei über die Inhalte entscheiden. Außerdem waren die formellen fachlichen Ansprüche an die Lehrkräfte relativ

gering. Häufig unterrichten interessierte Mathematiklehrkräfte. Mit der Einführung des Aufbaukurses und des Profilfachs IMP existiert nun ein verbindlicher Bildungsplan und ein gestiegener Anspruch an die Schulen und Lehrkräfte. In der Klassenstufe acht stehen somit zwei Wochenstunden, in den Klassenstufen neun und zehn jeweils eine Wochenstunde für die Informatik zur Verfügung. Zusätzlich überlappen zahlreiche der Mathematik zugeordnete Themen wie Kryptologie, Aussagenlogik und Graphen stark mit der Informatik. IMP soll dabei auf den Aufbaukurs Informatik, der seit 2016/2017 für alle siebten Klassen mit einer Wochenstunde verpflichtend ist, aufbauen.
Der am 29. März 2018 veröffentlichte Bildungsplan für die IMP legt weiterhin keine Programmiersprache fest, sondern äußert sich wie folgt [23]:

> Die Schülerinnen und Schüler implementieren Algorithmen in einer geeigneten Programmierumgebung und testen ihre Programme auf Fehler und die Ergebnisse auf Realitätsrelevanz.
>
> "Die Entscheidung für eine geeignete Programmiersprache beziehungsweise Programmierumgebung sollte in Kombination getroffen werden und nach Gesichtspunkten der altersangemessenen Vermittlung informatischer Konzepte erfolgen. Dabei ist der Auswahl der didaktischen Werkzeuge eine besondere Bedeutung beizumes- sen. Die Programmierumgebung sollte die Schülerinnen und Schüler bei der Eingabe und Strukturierung ihres Codes unterstützen, leichtes Auffinden und Beheben von Fehlern ermöglichen und möglichst auf die im Unterricht erforderlichen Funktionen beschränkt sein. Je nach eingesetzter Programmiersprache können objekt- orientierte Sprachelemente (zum Beispiel Methodenaufrufe bei Verwenden von Bibliotheken) notwendig sein. Diese können jedoch als spezielle Syntax aufgefasst und einfach nach Anleitung/Dokumentation verwendet werden. An eine Thematisierung der objektorientierten Programmierung – auch am Rande – ist nicht gedacht."

Es sind, wie aus dem Zitat hervorgeht, bei der Auswahl für den Unterricht die Sprache und die Programmierumgebung verknüpft. Diese Verknüpfung wird nicht nur im Bildungsplan, sondern auch in der Forschung zu Lehrwerkzeugen in der Informatik als relevant angesehen[1]. Da kaum geeignete Werkzeuge für eine unterrichtsgerechte Heranführung an Python existieren, ist diese Sprache im Moment noch als ungeeignet anzusehen.

2.2 Didaktischer Hintergrund

Eine Programmierumgebung ist für den Informatik Unterricht nicht zwingend notwendig. Sämtliche Konzepte könnten auch ohne das Schreiben einer einzelnen Codezeile vermittelt werden. Die pädagogische Forschung hat allerdings grundlegende Zusammenhänge

zwischen der Art des Unterrichts und dem Lernerfolg aufgedeckt. Aus diesem sehr umfangreichen Gebiet werden im Folgenden einige Ergebnisse präsentiert.

Es lässt sich 25% bis 50% der Varianz der Schüler bezüglich ihrer schulischen Leistungen auf die Motivation zurückführen [27]. Unterrichtsprinzipien, die die Motivation der SuS fördern sind demnach auch für den Lernerfolg als förderlich anzusehen. So bietet die aktive Auseinandersetzung mit einem Thema nach dem Erwartungs-mal-Wert Modell von Heckhausen motivationale Vorteile [28]. Dies gilt, wenn die SuS glauben, eigenständig in der Lage zu sein, die gesetzten Ziele zu erreichen. Diese Eigenständigkeit kann z.B. bei komplexeren Algorithmen nur durch Hilfestellung der Programmierumgebung sichergestellt werden.

Auch andere Pädagogen vertreten den positiven Einfluss der Selbsttätigkeit der SuS auf ihre Motivation: Deci und Ryan führen zum Beispiel das Erleben von Autonomie und Kompetenz als zwei ihrer drei zentralen situativen Gegebenheiten, die die Motivation fördern, auf [31]. Ebenso identifiziert Keller die Gelegenheit zur Anwendung erworbener Kenntnisse als Strategie der Motivation [32].

Neben der Motivation ist außerdem auch die Handlungsorientierung selbst leistungsfördernd. Dies hatte bereits Konfuzius mit dem bekannten Zitat

> "Erkläre mir, und ich vergesse. Zeige mir, und ich erinnere. Lass es mich tun, und ich verstehe."

vor tausenden Jahren festgestellt. Das Anwenden von eigenem Wissen zur Problemlösung, wie durch das tatsächliche Programmieren eines bekannten Algorithmus, lässt die SuS Kompetenz erfahren und vertieft den Lerneffekt weit über die reine Wissensvermittlung hinaus [29].

Das Unterrichtsprinzip Veranschaulichung spricht ebenfalls für die Verwendung einer Programmierumgebung, die abstrakte Objekte wie Graphen schematisch visuell darstellen kann. So können beide Hirnhälften angesprochen werden und sowohl die Motivation, als auch das Themenverständnis der SuS verbessert werden [29]. Klassischerweise wird hierfür ein reelles Objekt, Modell oder eine Veranschaulichung in einem anderen Medium verwendet. Dies ermöglicht die vorteilhafte Erfassung des Themas durch mehrere Sinne. Derartige Veranschaulichungen können ergänzend auch im Informatikunterricht angeboten werden. Der Vorteil der Visualisierung in der Programmierumgebung liegt darin, dass tatsächlich der Input der SuS selbst, statt einem allgemeinen Objekt anschaulich gemacht wird.

In der Literatur wird als wichtig angesehen, die SuS auch Fehler machen zu lassen [30].

Dieses *Trial and Error* Prinzip birgt aber die Gefahr der Demotivation durch Scheitern, insbesondere, da es oftmals nicht leicht ist, den Fehler zu identifizieren. Die Informatik bietet hier als Fach beste Voraussetzungen die SuS handeln zu lassen und bei jedem *Error* einen Hinweis auf die Fehlerquelle zu präsentieren. Durch eine entsprechend gestaltete Programmierumgebung kann somit der Anteil der demotivierten SuS gesteigert werden, während trotzdem die Vorteile der Selbsttätigkeit behalten werden.

Insgesamt ist die pädagogische Literatur der letzten vierzig Jahre im Hinblick auf die Nützlichkeit der Handlungsorientierung, Selbsttätigkeit und Veranschaulichung eindeutig. Kontroverse existiert nicht an der Kernaussage, sondern hauptsächlich bei der Erfassung des Motivationsbegriffes und der Sinnhaftigkeit verschiedener Motivationskategorien.

2.3 Programmiersprache

Wie bereits oben festgestellt, ist Python eine relevante, stark wachsende, high-level und syntaktisch einfache Programmiersprache. Daher wurde sie als eine gut für den Unterricht geeignete textuelle Programmiersprache identifiziert und die Programmierumgebung soll den SuS erlauben mit der Standard Syntax sowie den eingebauten Funktionen und Objekten von Python zu arbeiten.

2.4 Sprache

Die Sprache der Programmierumgebung sowie der Ein- und Ausgaben soll Deutsch sein. Zwar ist Englisch die Lingua franca der Informatik, allerdings findet der Unterricht auf Deutsch statt. Außerdem sollen SuS die sprachlich schwach sind nicht benachteiligt werden.

2.5 Abdeckung der IMP Inhalte

Die Programmierumgebung soll die Möglichkeit bieten einen großen Teil der Inhalte des Bildungsplans der IMP abzudecken. In der schulischen Realität ist aus Zeitgründen nicht davon auszugehen, dass jedes Thema in einer praktischen Übung durch die SuS programmiert werden kann. Trotzdem soll die Programmierumgebung ein umfassendes Werkzeug sein, durch welches nicht bei jedem neuen Thema ein Wechseln zwischen verschiedenen Programmen, Sprachen und neu zu erläuternden Benutzeroberflächen nötig wird. Die verschiedenen Themenbereiche des Bildungsplans wurden daher im Anhang II zusammengefasst, durch mögliche Programmierprojekte ergänzt und nach ihrer Eignung für ein Programmierprojekt bewertet. Die so erhaltene Rangfolge diente später als Grundlage für die Priorisierung verschiedener Themenmodule (vgl. Kapitel 4).

2.6 Vorkenntnis

Die Programmierumgebung muss nicht als der erste Kontakt der SuS mit einem Thema konzipiert sein. Tatsächlich sollte vor jeder praktischen Übung eines Themas die Erarbeitung im vorangegangenen Unterricht erfolgt sein. Die Vermittlung der dem jeweiligen Thema zugrundeliegenden Konzepte obliegt der Lehrkraft. Lediglich sämtliche notwendigen Informationen für die praktische Umsetzung des Themas müssen von der Programmierumgebung geliefert werden.

2.7 Spezielle Syntax

Die Programmierumgebung sollte es den SuS ermöglichen, ohne Kenntnisse spezieller Syntax Python zu programmieren. Der Sinn dieser Anforderung besteht darin, den SuS den Übergang von der schulischen Programmierumgebung zu in der Arbeitswelt verwendeten Programmierumgebungen zu erleichtern. Gleichzeitig wirkt die Kenntnis "echten"Code zu schreiben, motivationsfördernd [28]. Alternativer Code wie bei Kara oder farbige Codeblöcke wie bei Scratch (vgl.Kapitel 3) vermitteln nicht das Gefühl später anwendbare Kenntnisse zu erlernen. Grundsätzlich sollte daher der von SuS geschriebene Code als reiner Python Code ohne den Import spezieller Module ausführbar sein.

2.8 Themenfokus

Das Ziel eines jeden Programmierprojekts in der Programmierumgebung soll es sein, ein tieferes Verständnis des aktuellen Themas zu erlangen. Die Kenntnis der Programmiersprache ist lediglich als sekundär anzusehen. Daher sollten sämtliche Aspekte des Programmes, die nicht den gerade im Fokus stehenden Algorithmus betreffen, so weit wie möglich vereinfacht werden. Beispielsweise sollte bei der Umsetzung eines Graphenalgorithmus zur Tiefensuche nicht die Erstellung des Graphen im Vordergrund stehen. Stattdessen soll ausschließlich der Algorithmus auf einen bereits gegebenen Graphen angewendet werden.

2.9 Visualisierung

Die Visualisierung des von SuS geschriebenen Codes ist hilfreich für die Fehlerfindung und wirkt motivierend auf Schüler [11]. Beispielsweise verdeutlicht die animierte Abbildung eines Graphen den SuS die Funktionsweise verschiedener Graphenalgorithmen besser als ein rein in Listen- oder Matrixform gegebener und in der Konsole ausgegebener Graph.

2.10 Verlangsamte Ausführung

Für die Visualisierung, Fehlerfindung und das allgemeine Verständnis der Funktionsweise des eigenen Programmes ist eine verlangsamte Ausführung (oder zumindest Betrachtung) notwendig. Die SuS sollten die Möglichkeit haben ihr Programm in verschiedenen Geschwindigkeiten oder schrittweise nachzuvollziehen.

2.11 Hilfestellung und Umgang mit Fehleingaben

Bei der Arbeit mit bis zu 30 SuS ist die Aufmerksamkeit der Lehrkraft sehr geteilt. Daher sollen die SuS neben einer klaren Aufgabenstellung auch Erklärungen und Hilfestellungen für die Umsetzung der Aufgabe erhalten können. Beim Programmieren ist es für SuS oft nicht einfach, zu erkennen, wo ein Fehler liegt. Im besten Fall wird Ihnen die Fehlerart und die fehlerhafte Codezeile genannt. Bei einfachen Syntaxfehlern ist das ausreichend. Allerdings ist davon auszugehen, dass zahlreiche SuS den zugrundeliegenden Algorithmus noch nicht verinnerlicht haben. Logische Fehler sind demnach erst durch ein falsches Ergebnis oder z.B. Indexfehler erkennbar. Die Programmierumgebung sollte also einerseits spezifische Fehlermeldungen mit Lösungsansätzen auf deutsch ausgeben, andererseits für Logikfehler auch die Programmausführung bis zu dem Auftreten des Fehlers visualisieren.

2.12 Fortschrittssicherung

Der Fortschritt der SuS innerhalb eines Projektes aber auch über mehrere Projekte hinweg sollte gesichert und vom Lehrer und den SuS eingesehen werden können. So soll ein Arbeiten über mehrere Schulstunden möglich sein. Außerdem dient die Fortschrittsspeicherung den SuS als Motivation und der Lehrkraft als Bewertungsgrundlage.

2.13 Verfügbarkeit

Die Programmierumgebung soll möglichst barrierefrei nutzbar sein. Administratorrechte, Installationen oder bereits eingerichtete Python Umgebungen sollten nicht benötigt werden. Eine Nutzung des Programms für SuS von privaten Geräten soll ebenfalls möglich sein.

2.14 Datenschutz

Die Datenschutzbestimmungen an baden-württembergischen Gymnasien sind enorm streng. Es dürfen keinerlei personenbezogene Daten der SuS außerhalb des Verwaltungsnetzes der Schule gespeichert werden [2]. Da dies leistungsbezogene Daten beinhaltet ist eine denkbare personalisierte Fortschrittsspeicherung einer Programmierumgebung nur unter bestimmten Einschränkungen möglich. So dürften sich die SuS bei einem individuellen

Login nur mit anonymisierten Nutzerdaten einloggen und der Schlüssel dürfte nur im Verwaltungsnetz der Schule verfügbar sein.

2.15 Sicherheit vor bösartigem Code

Sobald die SuS tatsächlichen Python Code ausführen können besteht die Möglichkeit, bösartigen Code auszuführen. Die SuS müssen dies nicht beabsichtigt haben, selbst ungewollte Endlosschleifen, die den PC blockieren können als bösartig angesehen werden. Dies ist je nach Aufbau der Programmierumgebung mehr oder weniger relevant. In einem Schulnetz, das die einzelnen Schüler-PCs von einem unveränderlichen Image startet kann der Code weniger Schaden anrichten als bei einer Weblösung bei der Schülercode auf der Serverseite ausgeführt wird. Dementsprechend ist die Programmierumgebung so zu gestalten, dass durch die SuS kein permanenter Schaden angerichtet werden kann der nicht schlimmstenfalls durch einen Neustart des PCs behoben werden kann.

3 Analyse existierender Lösungen

Dieses Kapitel stellt eine Auswahl der am weitesten verbreiteten Programmierumgebungen für SuS oder Lernende im Detail vor. Die größten Vor- und Nachteile der Programmierumgebungen werden dargestellt und die Eignung für den IMP Unterricht geprüft. Dabei werden einzelne, für eine eigens erstellte Programmierumgebung interessante Merkmale hervorgehoben. Bei der Auswahl der hier präsentierten Werkzeuge wurden verschiedene Quellen gesichtet [6, 7, 8], sowie ein erfahrener Informatiklehrer, Bernd Moll, und der stellvertretende Sprecher der Informatiklehrerinnen und -lehrer in Baden-Württemberg (ILLBW) der Gesellschaft für Informatik, Urs Lautebach, befragt. Oftmals wurden sonst relevante Werkzeuge vernachlässigt, da eine ähnliche Alternative bereits aufgeführt ist. So existieren z.B. für Scratch und Code.org viele ähnliche Alternativen. Außerdem wurden zahlreiche, auch in der Schule Anwendung findende, Umgebungen vernachlässigt, die spezielle Hardware erfordern wie z.B. Arduino oder Lego Mindstorms. Jede der aufgeführten Alternativen wurde vom Autor installiert und tatsächlich getestet.
Die Kompatibilität mit den Anforderungen aus Kapitel 2 ist in Tabelle 1 zusammengefasst.

Kriterium (Kapitel)	Code.org	Pythontutor.com	Codecademy	Turtle	Greenfoot	Kara	Scratch	BlueJ	Klassische Programmierumgebung
2.3 Prog. Sprache	Graf.	Py	Versch.	Py	Java	Py	Graf.	Java	Versch.
2.4 Sprache	D	E	E	D	D	D	D	D	D
2.5 IMP Inhalte	-	-	-	-	0	-	-	0	0
2.7 Spez. Syntax	+	-	-	+	+	+	+	-	-
2.8 Themenfokus	-	-	-	-	-	-	-	-	-
2.9 Visualisierung	+	0	-	+	+	+	+	-	-
2.10 Verlangsamung	+	+	-	+	+	+	-	-	0
2.11 Hilfestellung	+	-	+	0	0	+	0	0	0
2.12 Fortschrittssich.	+	-	+	+	+	+	+	+	+
2.13 Verfügbarkeit	+	+	+	+	+	+	+	+	+
2.14 Datenschutz	-	+	-	+	+	+	+	+	+
2.15 Sicherheit	+	+	+	-	-	+	+	-	-

Tabelle 1: Vergleich einer Auswahl von alternativen Programmierumgebungen

3.1 Code.org

Code.org ist eine unter anderem von Microsoft und Facebook gegründete non-profit Organisation mit dem Ziel weltweit Schülern das Programmieren beizubringen. Programmiert wird ausschließlich mit Programmblöcken, die wie Puzzleteile aneinandergefügt werden (vgl. Abbildung 1). Lehrkräfte können Lerngruppen anmelden, verwalten und ihren Fortschritt über die Website kontrollieren (vgl. Abbildung 2).

Diese grafische Programmierung minimiert demotivierende Syntaxfehler und veranschau-

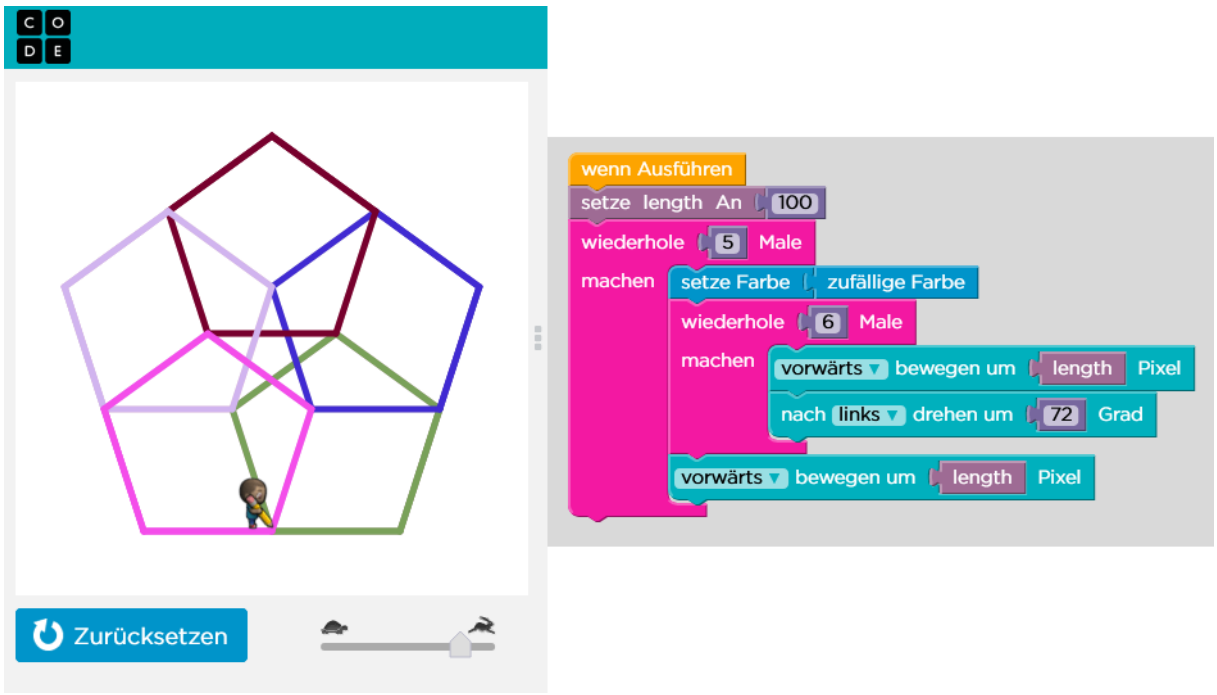


Abbildung 1: Grafische Programmieroberfläche von Code.org [4]. Zu erkennen ist links der Animationsbereich und er Regler für die Geschwindigkeit, rechts die Codeblöcke.

licht den Code. Code.org nutzt außerdem aufwändige Animationen um den Programmablauf darzustellen. Dies und eine große Auswahl von verschiedenen Schwierigkeiten und Lerngeschwindigkeiten macht Code.org zu einer guten Wahl für den Einstieg in die Grundkonzepte der Programmierung, wie If-Else Abfragen, Schleifen, Variablen und Funktionen. Für eine fortschrittlichere Anwendung wie IMP fehlt die Verknüpfung mit einer echten Programmiersprache und die Auseinandersetzung mit den speziellen Inhalten der IMP. Die Fortschrittsverfolgung der SuS gibt der Lehrkraft eine sehr gute Übersicht und Bewertungsgrundlage, allerdings ist die Anwendung von Code.org nach derzeitigen Datenschutzbestimmungen an baden-württembergischen Gymnasien nicht zulässig.

3.2 Pythontutor

Pythontutor ist ebenfalls eine webbasierte Programmierumgebung. Pythontutor bietet die Möglichkeit zu jedem Zeitpunkt der Programmausführung den global Namespace darzu-

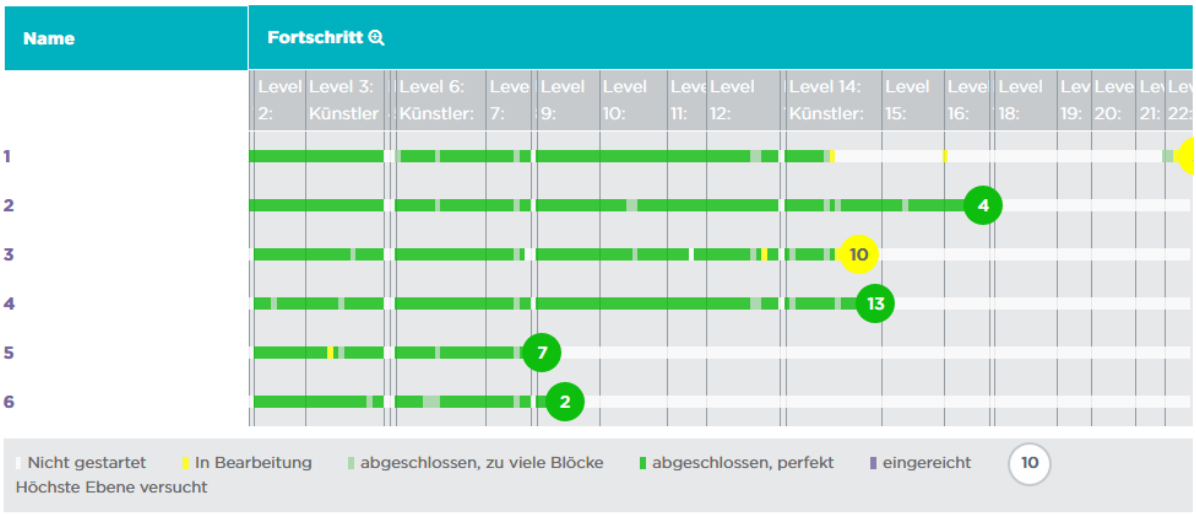


Abbildung 2: Fortschritt einer Lerngruppe am Erasmus-Gymnasium Denzlingen[4]. Die Schülerkürzel sind unkenntlich gemacht. Hellgrüne Felder sind gelöste, aber nicht mit der minimalen Blockzahl gelöste Aufgaben. Gelbe Felder entsprechen nicht gelösten Aufgaben.

stellen. Dafür wird der Pythoncode vom Client in JavaScript umgewandelt, ausgeführt und der Stand der Objekte über Traces ausgelesen. Obwohl also Pythoncode eingegeben wurde, kann der tatsächlich ausgeführte JavaScript Code keinen Schaden anrichten. Während Pythontutor zum Debuggen bei kleinen Programmen Verwendung finden kann ist er aufgrund der fehlenden Anleitung für die Schule weniger gut geeignet. Zudem ist die Website für Schüler nicht ansprechend gestaltet, es ist z.B. nicht möglich einen Graphen grafisch darzustellen. Die konsequente Darstellung aller Variablenbesetzungen wie in Abbildung 3 ist allerdings ein Konzept, dass für eine gute Programmierumgebung hilfreich sein kann.

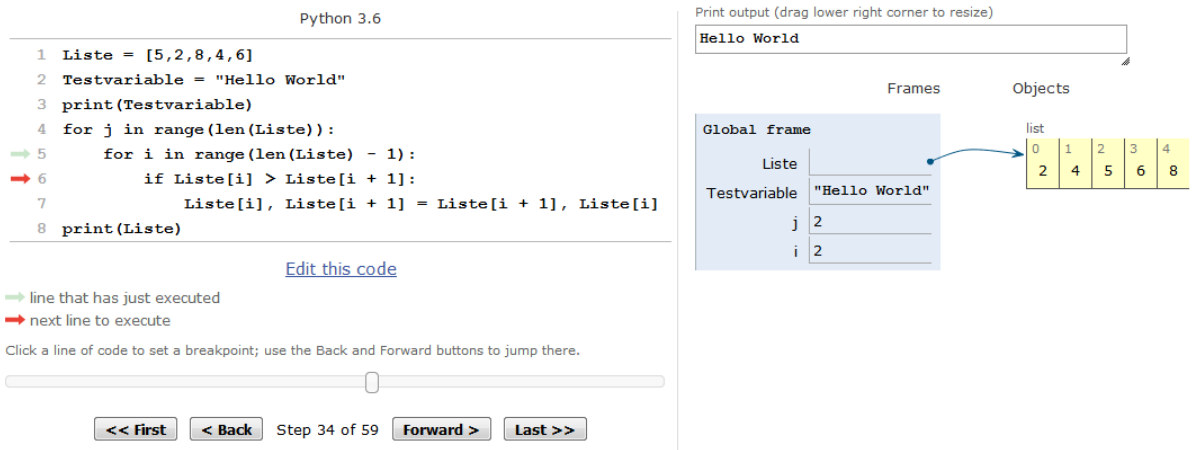


Abbildung 3: Visualisierung von Bubblesort mit Pythontutor[14]. Links der Code mit Anzeige der gerade ausgeführten Zeile und der Fortschrittsanzeige. Rechts die Konsole und der Zustand aller verwendeter Variablen

3.3 Online Kurse am Beispiel codecademy

Zahlreiche Anbieter wie codecademy, Codeschool oder Treehouse bieten Kurse in Python und anderen Programmiersprachen an, in denen vom Nutzer aktiv Code nach Anleitung geschrieben wird. Diese Angebote sind fast ausschließlich auf Englisch verfügbar. Sie folgen strikt vorgegebenen Lernplänen und richten sich in den meisten Fällen an Anfänger mit dem Ziel, die grundlegende Syntax zu vermitteln. Während hier eine gute Anleitung und Fortschrittskontrolle gegeben sind, überschneidet sich der strikte vorgegebene Plan kaum mit den IMP Inhalten. Die Gamifizierung des programmieren Lernens durch Achievements wie die ”25 Exercises Medaille”auf codecademy.com ist ein interessanter Aspekt als zusätzliche Motivation für die SuS. Auch die gleichzeitige Darstellung von Aufgabe, Hilfestellung und Code Ausgabe wie in Abbildung 4 bietet einen guten Ansatz für eine eigens erstellte Programmierumgebung. Diese und ähnliche Umgebungen werden allerdings von kommerziellen Anbietern erstellt, die in der Schule nicht beworben werden dürfen und die außerdem die Angabe persönlicher Daten voraussetzen.

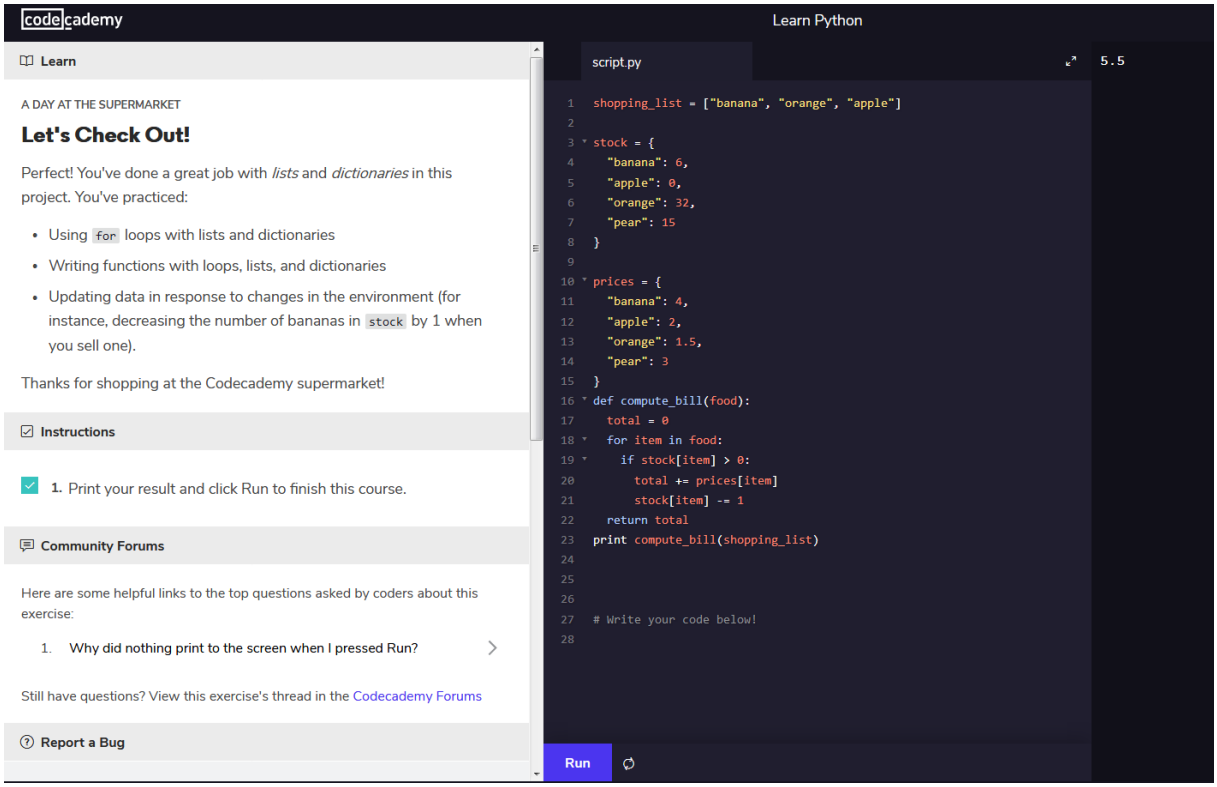


Abbildung 4: Nutzeroberfläche von codecademy mit Anleitung links, Code in der Mitte und Ausgabe rechts[5].

3.4 Turtle Modul

Es handelt sich bei Turtle nicht um eine Programmierumgebung, sondern um ein Modul für Python, das mit dem Zweck, Kindern Programmieren beizubringen entwickelt

wurde. Es kann in diesem Modul ein Stift (turtle) mit Befehlen wie `forward(100)` oder `right(90)` gesteuert werden. Mit relativ einfachen Schleifen können so interessante und eindrucksvolle Muster entstehen. Obwohl Turtle keine eigene Programmierumgebung ist, wird es hier trotzdem aufgenommen, da zahlreiche Bücher und Anleitungen existieren, die auf die Vermittlung von Konzepten der Informatik mit diesem Modul abzielen. In Verbindung mit solchen Anleitungen sind also einige Anforderungen erfüllt, allerdings eignet sich das Modul nicht für die komplexeren Themen der IMP. Es wird hier dennoch aufgeführt, da das Ziel der eigens entwickelten Programmierumgebung bisher nicht die Vermittlung von grundlegender Syntax von Python ist. Mit dem standardmäßig in Python verfügbaren Turtle Modul wäre eine einfache Implementierung zur Vermittlung dieser grundlegenden Konzepte mit geringem Aufwand möglich. Somit müsste mit einer entsprechenden Anleitung kein Vorwissen über Python mehr vorausgesetzt werden. Abbildung 5 zeigt das Resultat eines Beispielcodes.

```
from turtle import *

screen = Screen()
screen.bgcolor("white")
lea = Turtle()
lea.speed(9)
lea.color("black")

#Mandala
for j in range(10):
    for i in range (360):
        lea.forward(2)
        lea.right(1)
    lea.right(36)

screen.exitonclick()
```

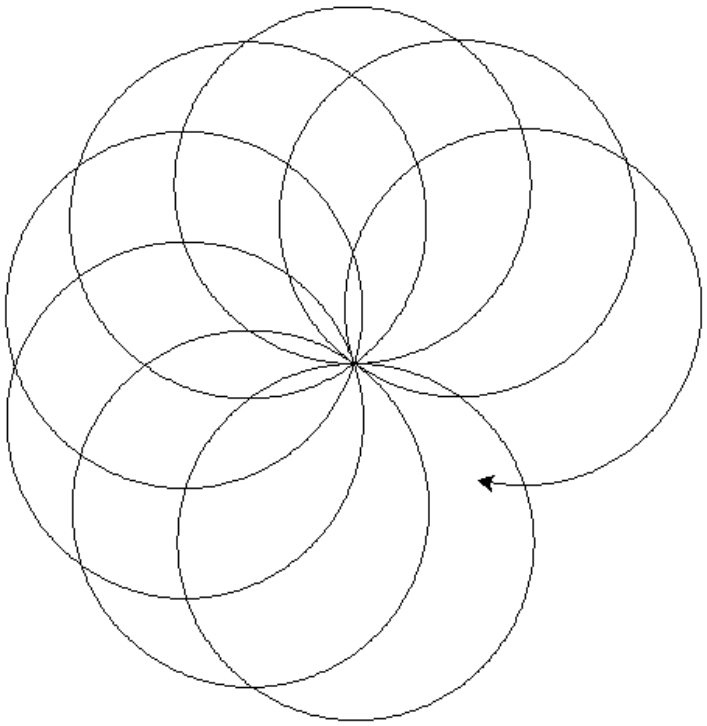


Abbildung 5: Exemplarischer Code und Visualisierung von Turtle.

3.5 BlueJ

BlueJ ist eine Entwicklungsumgebung für Java, die sich auf die Vermittlung von Konzepten der objektorientierten Programmierung spezialisiert. BlueJ kann als vollständige Entwicklungsumgebung auch für komplexe Java Projekte genutzt werden, ist also theoretisch in der Lage sämtliche Themen der IMP abzudecken, auch wenn diese nicht im

Fokus stehen. Anleitungen für SuS und Lehrkräfte sind kostenpflichtig. BlueJ ist zweifelsohne ein mächtiges Werkzeug mit dem einer motivierten Lehrkraft viele Möglichkeiten offen stehen. Gleichzeitig ist der Zeitaufwand und Betreuungsaufwand ohne beträchtliche Vorarbeit der Lehrkraft sehr hoch. Es eignet sich demnach eher für große, längerfristige Projekte beispielsweise in der Jahrgangsstufe. Für die Vermittlung der IMP Inhalte, bei der die eigentliche Programmiertätigkeit im Hintergrund steht, ist der Aufwand zu hoch. Dies gilt insbesondere, da die objektorientierte Programmierung, deren Vermittlung der Hauptnutzen von BlueJ ist, nicht im IMP Bildungsplan vorgesehen ist. Abbildung 6 zeigt ein Beispielprojekt, in dem von Grund auf ein Taschenrechner programmiert wird. Hier ist auch zu erkennen, dass das Klassendiagramm erst Sinn macht, wenn die Komplexität des Programmes bereits recht groß ist.

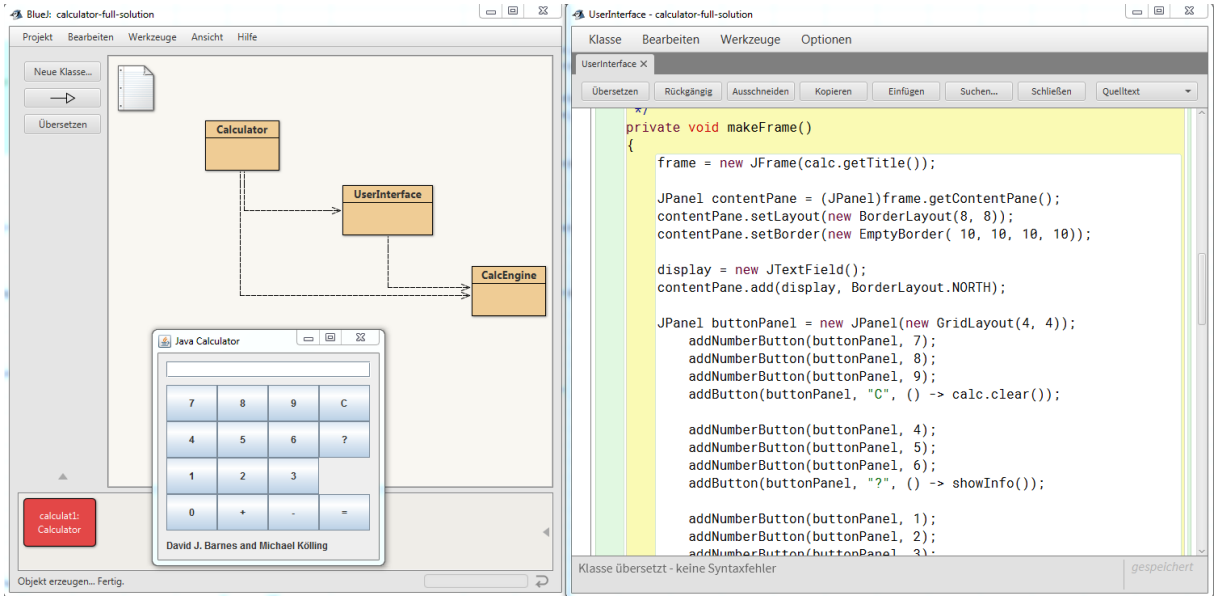


Abbildung 6: Klassendiagramm (links), Code (rechts) und Resultat eines Taschenrechner-Programmes (links unten) in BlueJ[12]

3.6 Greenfoot

Greenfoot ist eine auf BlueJ basierende Programmierumgebung, die die Programmierung von 2D Anwendungen enorm erleichtert. Zahlreiche vorgefertigte Welten und Grafiken machen es möglich, mit geringem Aufwand sogenannte Actor Objekte zu erstellen und ihr Verhalten als Instanz in der 2D Welt zu Programmieren. Im Fokus steht hierbei, wie auch bei BlueJ, das Konzept der objektorientierten Programmierung mit Klassen, Instanzen, Methodenaufrufen und Vererbung. Dies wird in Abbildung 7 deutlich. Besonders interessant ist hierbei die gelungene, unterbrechbare und für Schüler motivierende Visualisierung, die oft einen Spielcharakter hat, sowie der Object Inspector, der zu jedem Zeitpunkt die Eigenschaften der verschiedenen Objekte anzeigen kann. Gleichzeitig machen die selben Nachteile wie bei BlueJ Greenfoot für die Anwendung im Profilfach IMP ungeeignet.

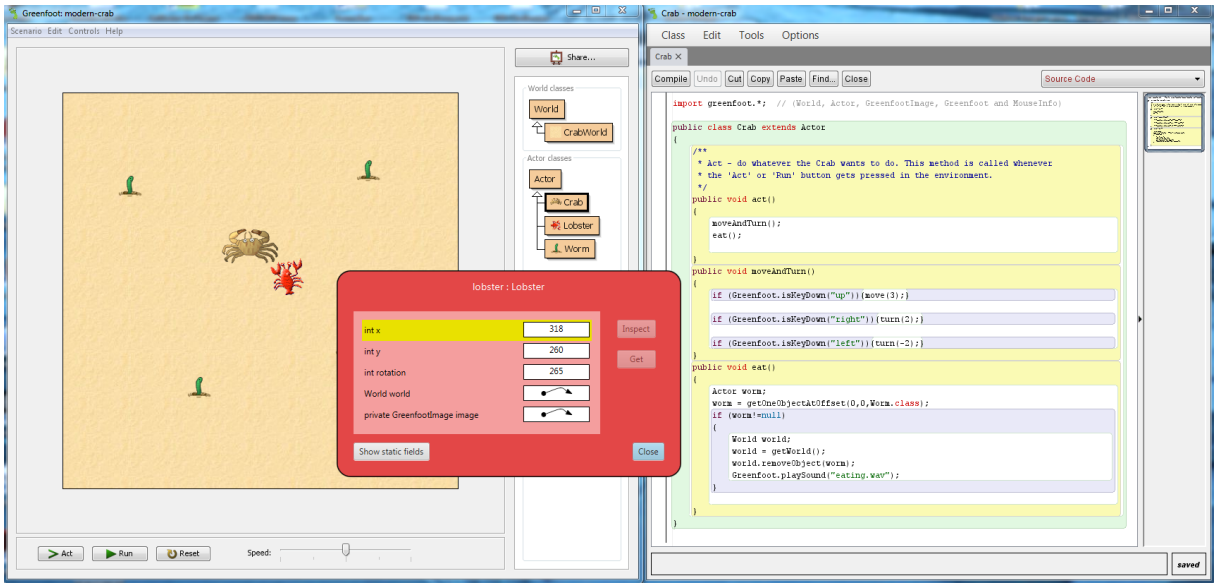


Abbildung 7: Visualisierung (links), Object Inspector (mitte) und Code (rechts) in Greenfoot[9]

3.7 Kara

Kara ist eine ursprünglich für Java entwickelte Programmierumgebung für das Programmieren mit endlichen Automaten aber auch Java Code [11]. Eine Abwandlung davon, PythonKara, unterstützt Python. Die Benutzeroberfläche von PythonKara ist in Abbildung 8 dargestellt. Es sind die Elemente zur Anleitung, Visualisierung und dem Code zu erkennen. Es werden Aufgaben mit Anleitungen angeboten, die immer komplexeres Verhalten von Kara - dem vom eigenen Code gesteuerten Marienkäfer - verlangen. Auch bei dieser Programmierumgebung ist der Umfang der untersuchbaren Datenstrukturen und Algorithmen begrenzt. Die Visualisierung macht das programmierte Verhalten deutlich, allerdings werden Abfragen wie `treeLeft()` nicht sichtbar gemacht. Da außerdem nicht hervorgehoben wird, welche Programmzeile gerade ausgeführt wird, ist die Visualisierung deutlich verbesserungswürdig. Diese Mängel flossen als Anregung in die eigens erstellte Programmierumgebung ein.

3.8 Scratch

Die webbasierte, oder auch offline verfügbare, interaktive grafische Programmierumgebung Scratch ist in Schulen gut bekannt. Sie eignet sich vor allem, um geskriptete Animationen oder einfache Spiele zu erstellen. Scratch ist optisch sehr kinderfreundlich gestaltet, überflutet aber Anfänger mit Optionen (vgl. Abbildung 9). Aus persönlicher Erfahrung am Erasmus-Gymnasium in Denzlingen ließ sich feststellen, dass nur unter strengster Anleitung neue Konzepte der Informatik mit Scratch vermittelt werden können. Eine freiere Auseinandersetzung der SuS mit Scratch führt zu simplen, aber langen Befehlsfolgen. Zwar können grundlegende Konzepte ohne die Hürde von tatsächlichem Code mit feh-

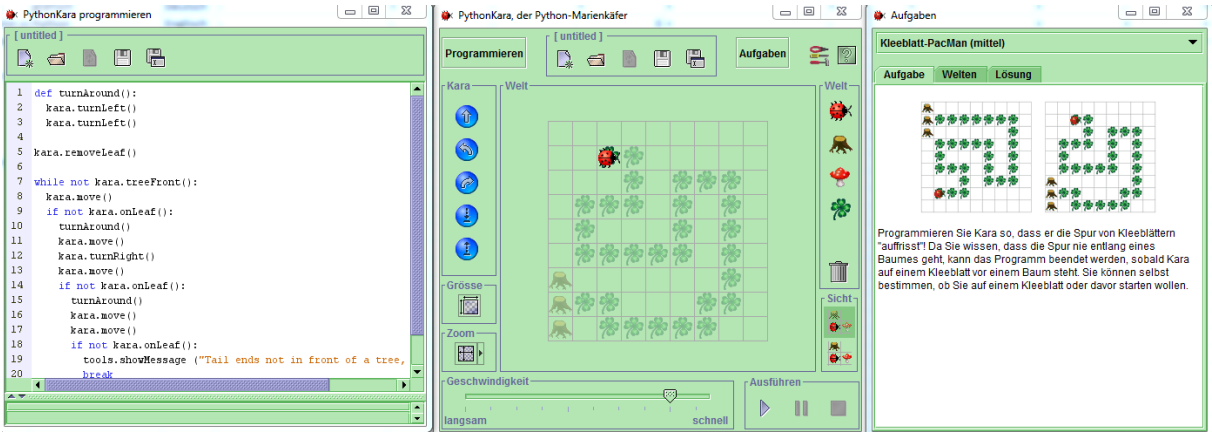


Abbildung 8: Code (links), Visualisierung (mitte) und Anleitung (rechts) in Kara[10]

leranfälliger Syntax vermittelt werden, darüber hinaus ist aber eine sinnvolle Nutzung mit einer gesamten Schulklasse schwierig. Gleichzeitig konnte beobachtet werden, dass einzelne SuS durch die ansprechende Oberfläche enorm motiviert und auch kreativ waren. Dieser Aspekt ist schwer mit der Vermittlung von spezifischen Inhalten der IMP vereinbar, hat aber das Potential die Schülermotivation stark zu fördern.

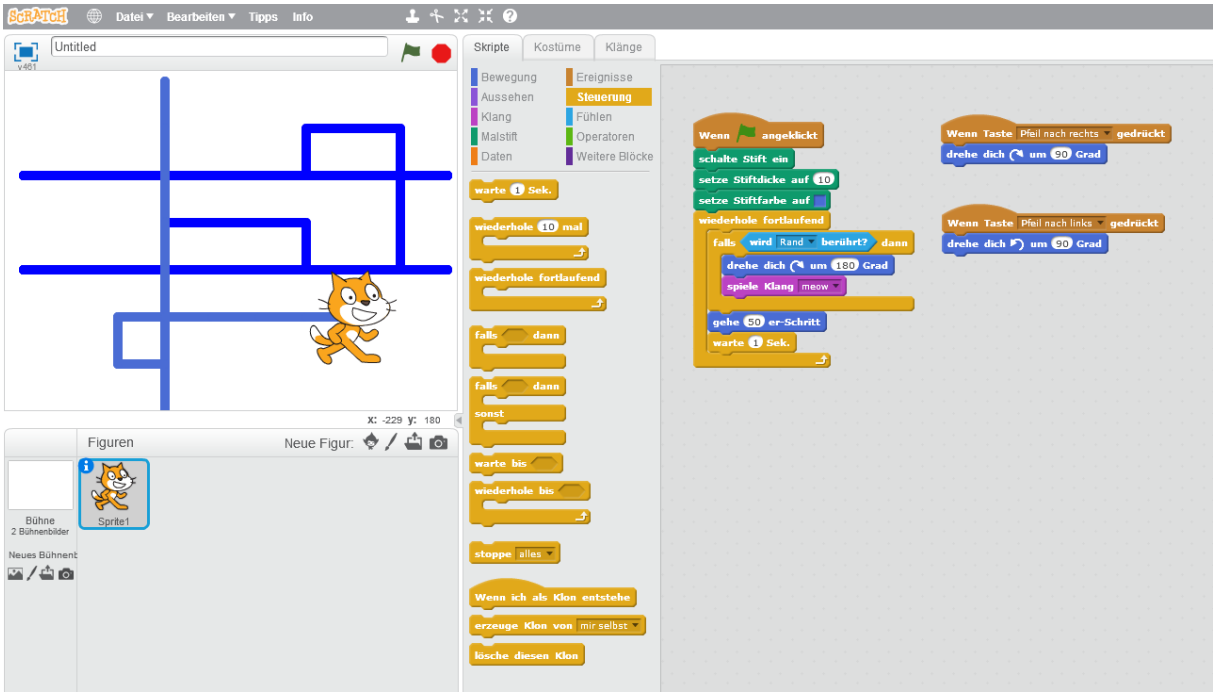


Abbildung 9: Visualisierung (links oben), verfügbare Codeblöcke und Kategorien (mitte) sowie Objektcode(rechts) in Scratch [13]

3.9 Andere Programmierumgebungen

Mit dem großen Angebot von Texteditoren über professionelle Entwicklungsumgebungen stehen der Lehrkraft neben den speziell für die Bildung ausgelegten Programmierumge-

bungen auch andere Möglichkeiten offen. Grundsätzlich lässt sich für diese Möglichkeiten die fehlende Anleitung, Einschränkung, Übersichtlichkeit und Visualisierung als Nachteil feststellen. Für größere Programmierprojekte, speziell in der Jahrgangsstufe, sind diese Angebote geeignet. Das Ziel, ausschließlich ein spezielles Thema durch praktische Anwendung verständlich zu machen, lässt sich mit ihnen allerdings nur mit großem Zeitaufwand erreichen. Trotzdem sind viele Aspekte wie Syntaxhighlighter, Codevervollständigung, Fehlererkennung etc. auch für SuS nützlich, sofern sie das GUI nicht überladen.

3.10 Andere Software

Für zahlreiche Themen der IMP wie z.B. Netzwerke oder Kryptografie sind zwar Programmieraufgaben für die SuS denkbar, aber nicht die beste Methode um Zusammenhänge und Funktionsweisen zu vermitteln (vgl. Anhang II). Hier existieren zahlreiche analoge Hilfsmittel und Softwaretools wie Filius [15] für die Simulation von Netzwerken oder Cryptool [16] für die Ver- und Entschlüsselung mit verschiedenen Chiffren. Da eine eigene Programmierumgebung für diese Themen nicht attraktiv ist, werden in dieser Arbeit auch diese im Unterricht verwendeten Tools nicht weiter erläutert.

3.11 Fazit

Aus der vorangehenden Auflistung wird deutlich, dass ein großes Angebot an Lernsoftware existiert. Aus Tabelle 1 geht allerdings auch hervor, dass keine der bekanntesten oder in der Schule bisher verwendeten Programmierumgebungen für die IMP Themen geeignet sind. Stattdessen liegt der Fokus fast immer entweder auf grundlegenden Konzepten des Programmierens wie Variablen und Schleifen, oder auf der objektorientierten Programmierung. Dieser Trend bei Programmierumgebungen für den Unterricht wurde auch in einer 2013 durchgeführten Metastudie festgestellt: von 45 untersuchten Papern, die Unterrichtstools untersuchen, beschäftigten sich 43 mit dem Einstieg in die Programmierung[1]. Lediglich in Greenfoot, BlueJ und klassischen Programmierumgebungen wäre eine Auseinandersetzung mit den IMP Themen überhaupt möglich. Selbst bei diesen Umgebungen fehlt aber die passende Visualisierung und der Fokus auf das zu behandelnde Thema. Während es also beispielsweise möglich ist, SuS ein Graphenobjekt erstellen zu lassen, auf dem dann eine Tiefensuche stattfinden kann, wäre diese Aufgabe weder angemessen visualisiert, noch in einem akzeptablen Zeitrahmen möglich. Es kann somit eindeutig eine Lücke im Werkzeugkasten der Informatiklehrkräfte festgestellt werden. Den SuS fehlt die Möglichkeit, selbst die in der IMP vermittelten Algorithmen und Konzepte umzusetzen.

4 Die Programmierumgebung

Wie im Einführungskapitel erläutert ist das Ziel dieser Arbeit eine funktionsfähige Programmierumgebung mit der Algorithmen der IMP von SuS praktisch umgesetzt werden können. In Kapitel 4.1 wird diese Programmierumgebung und ihre Funktionen vorgestellt. Kapitel 4.2 gibt anschließend einen Einblick in die technischen Hintergründe.

4.1 Funktionen

Abbildung 10 zeigt die Benutzeroberfläche der Programmierumgebung. Abbildung 11 zeigt dieselbe Oberfläche und hebt die verschiedenen Arbeitsbereiche hervor.

Nachdem im Wahlbereich für Kategorie und Aufgabe(2) ein Modul ausgewählt wur-

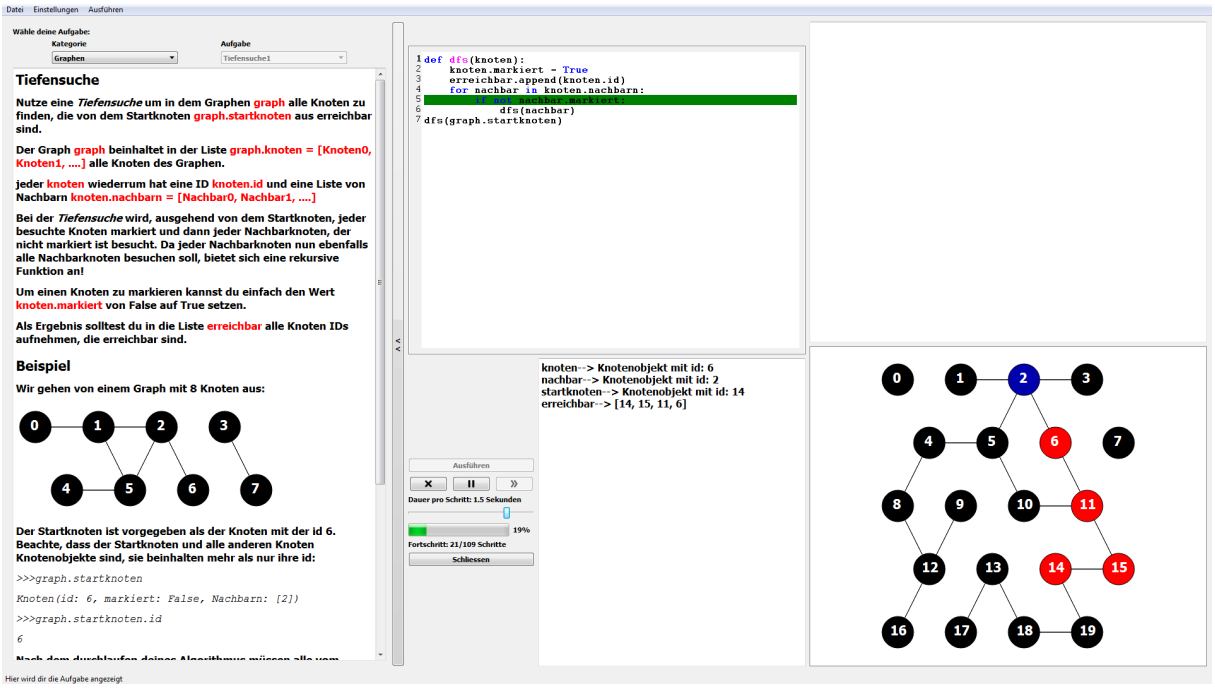


Abbildung 10: Benutzeroberfläche der Programmierumgebung

de, wie in diesem Beispiel die Tiefensuche in der Kategorie Graphen, wird den SuS im Bereich Aufgabenbeschreibung(4) die zugehörige Anleitung im HTML Format angezeigt. Diese Anleitung beinhaltet die Aufgabenbeschreibung, eventuelle Hilfestellung und die bereits zur Verfügung gestellten Objekte, die durch den SuS manipuliert werden sollen. In diesem Beispiel wird ein Graph als abgewandelte Adjazenzliste zur Verfügung gestellt.

Die SuS können nun ihren Code in den Codeeditor(5) eintragen. Hier wird vom Syntaxhighlighter der Code lesbarer gemacht. Syntaxfehler können vor dem Ausführen des Programmes noch nicht hervorgehoben werden (vgl. Kapitel4.2.5). In diesem Bereich ist während der Ausführung außerdem die aktuell ausgeführte Zeile grün hinterlegt. Im Falle

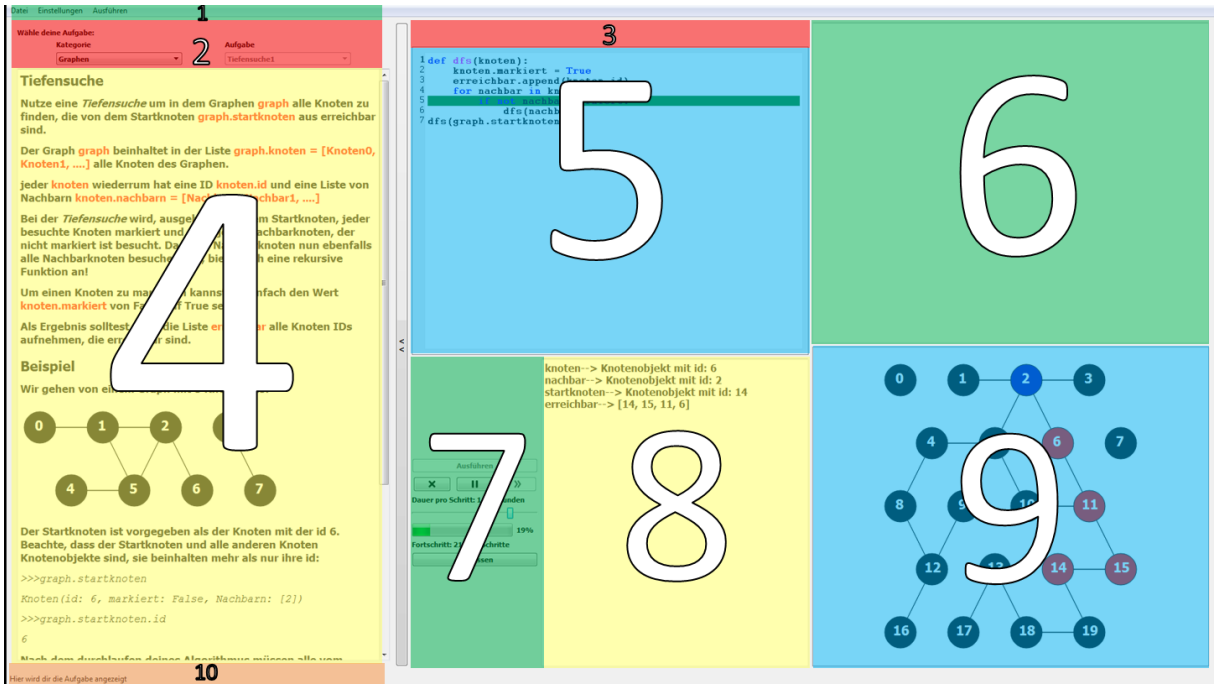


Abbildung 11: Arbeitsbereiche der Benutzeroberfläche: (1)Menüleiste, (2)Wahlbereich für Kategorie und Aufgabe, (3)Vorgegebene Objekte und Variablen, (4)Aufgabenbeschreibung, (5)Codeeditor, (6)Konsole, (7)Steuerungsbereich, (8)Ausgabebereich der lokalen Variablen, (9)Animationsbereich, (10)Statusleiste

eines Errors wird die betroffene Codezeile rot hinterlegt.

Im Steuerungsbereich(7) können die SuS die Ausführungsgeschwindigkeit über den Schieberegler anpassen. Während der Codeausführung können sie das Programm stoppen, schrittweise weiter ausführen oder abbrechen. Zudem wird ein in Prozent und absoluten Schritten angegebener Fortschrittsbalken angezeigt.

Der Variablenbereich stellt sämtliche im SuS Code deklarierten Variablen mit Namen und Inhalt dar. Zusätzlich wird der Zustand von im Modul vorgegebenen Objekten angezeigt.

Die Konsole(6) druckt die `print` Befehle der SuS. Zudem werden je nach Fehlerart spezifische eigens generierte Errormeldungen mit Lösungshinweisen auf deutsch und mit der entsprechenden fehlerhaften Codezeile ausgegeben (vgl. Kapitel 4.2.9). Je nach Modul können hier zusätzliche Ausgaben erscheinen, die z.B. das erfolgreiche Lösen der Aufgabe bestätigen.

Der Inhalt des Animationsbereiches(9) hängt nun ausschließlich vom gerade geöffneten Modul ab. Vor dem ersten Ausführen ist er immer leer. In diesem Beispiel ist ein zufällig erstellter Graph zu sehen, auf dem die Tiefensuche läuft. Auf die genauen Inhalte wird

bei der nachfolgenden Beschreibung der Module eingegangen.

In der Menüleiste(1) können die SuS ihren Code speichern oder gespeicherten Code laden, sowie zwischen Vollbild und Fenstermodus wechseln. Die Statusleiste(10) zeigt Informationen über den unter dem Mauszeiger befindlichen Bereich an.

4.1.1 Modul Bubblesort und Insertionsort

Die Module Bubblesort und Insertionsort sind bis auf die Aufgabenbeschreibung fast identisch. Es wird den SuS eine mit Zufallszahlen zwischen 0 bis 9 gefüllte Liste zur Verfügung gestellt. Im Animationsbereich wird die Liste als eine Reihe von mit dem Index beschrifteten Schubladen dargestellt (vgl. Abbildung 12). Jeder Aufruf und jede Änderung eines Elementes der Liste *öffnet* eine Schublade. Somit wird dem Schüler verdeutlicht, dass der Algorithmus zu keinem Zeitpunkt eine Übersicht über die komplette Liste hat. Um das Ändern eines Eintrages zu verdeutlichen wird die vorhandene Ziffer in einer Animation aus der Schublade *herausgeworfen* und die neue *eingelegt*. Am Ende der Durchführung werden alle Schubladen geöffnet gezeigt und die SuS erhalten die Rückmeldung, ob die Liste korrekt sortiert wurde über die Konsole.

Abbildung 12: Visualisierung der mit Bubblesort zu sortierenden Liste. Gezeigt ist der Vergleich der Elemente 3 und 4 der Liste. Da der Vergleich in einer Codezeile stattfindet, sind beide Schubladen gleichzeitig geöffnet.

4.1.2 Modul Listen

Das Listenmodul ist als beispielhafte Einführung in die Syntax von Listen gedacht. Die Aufgabenbeschreibung beinhaltet mehrere kurze Aufgaben, nach denen eine vorgegebene aber zufällige Liste manipuliert werden soll. Der erfolgreiche Abschluss jeder Unteraufgabe wird dabei nach Ausführung des Programmes bekanntgegeben.

4.1.3 Modul Turtle

Es wird mit dem Turtle Modul eine dem Python Modul Turtle sehr ähnliche Umgebung zur Verfügung gestellt. Mit dem Turtle Modul können durch von der Lehrkraft vorgegebene Aufgaben die Konzepte Variablen, Funktionen und Schleifen sehr gut und anschaulich

eingeführt werden. Da diese Themen bereits in Klassenstufe sieben Thema sind, sollten sie den SuS konzeptionell nicht schwer fallen. Das Modul ist also zur Wiederauffrischung und zur Vermittlung der Python Syntax geeignet.

4.1.4 Graphenmodule

Bei den Graphenmodulen werden ungerichtete und gerichtete Graphen generiert. Die Aufgabenbeschreibung stellt den Graphen und die Knoten vor, sowie die ihnen zugehörigen Attribute, wie z.B. boolesche Variable `markiert` bei Knoten. Die SuS schreiben ein Programm, das, je nach Aufgabe, über Tiefen- oder Breitensuche alle von einem Startknoten erreichbaren Knoten oder einen Pfad zu einem Zielknoten finden. Der Animationsbereich stellt dabei den Graphen dar und ist in Abbildung 13 zu erkennen. Wird ein Knoten markiert, blinkt er kurz rot und verbleibt dann rot. Bei einer Abfrage, ob ein Knoten markiert ist oder nicht, blinkt er kurz blau und verbleibt dann in der Ausgangsfarbe.

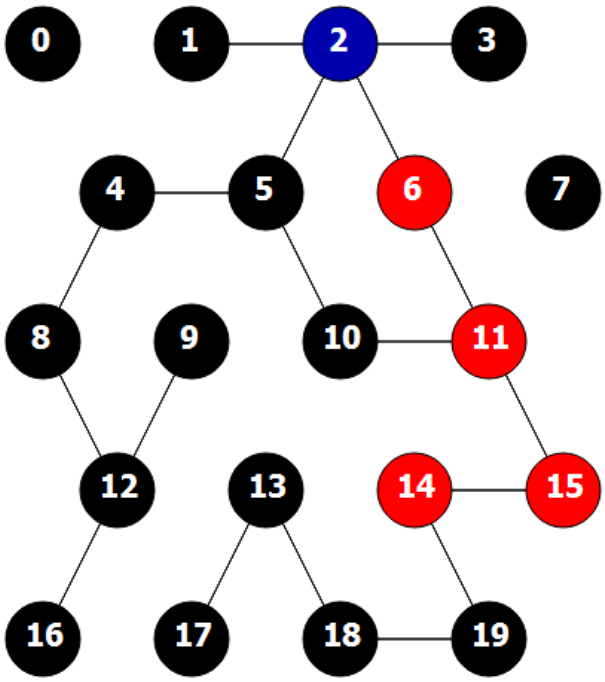


Abbildung 13: Animationsbereich der Graphenmodule mit besuchten Knoten in rot und Knoten, deren Zustand gerade abgefragt wird in blau.

4.1.5 Modul Freies Programmieren

In diesem Modul wird nichts animiert und es existieren keine vorgegebenen Objekte. Es dient dem Experimentieren oder durch Anleitung der Lehrkraft zum Erlernen der Python-Syntax.

4.1.6 Modul Prüfziffern

Die inhaltsbezogene Kompetenz *Verfahren zur Fehlererkennung (unter anderem Prüfsumme)*[23] kann in diesem Modul vermittelt werden. Es wird eine Liste mit ISBN Nummern zur Verfügung gestellt, in der zufällige Einträge zufällige Fehler beinhalten. Diese fehlerhaften Einträge sollen von den SuS erkannt und aussortiert werden. Dafür ist in der Aufgabenbeschreibung das Prüfverfahren für ISBNs aufgeführt. Es wird in diesem Modul nichts visualisiert.

4.2 Entwicklungsprozess und Umsetzung

Bei der Entwicklung der Programmierumgebung mit den oben aufgeführten Anforderungen ergeben sich zahlreiche Herausforderungen. Diese und die gewählten Lösungsansätze werden in diesem Kapitel aufgeführt und erläutert.

4.2.1 Weblösung

Ein naheliegender Ansatz für die Programmierumgebung ist eine im Browser nutzbare, und somit für die SuS von überall verfügbare Implementierung. Hierbei stehen grundsätzlich zwei Optionen zur Wahl: eine server- oder clientseitige Umsetzung.

Die serverseitige Umsetzung ist von Beginn an als problematisch anzusehen, da hier unbekannter, von SuS geschriebener Python Code auf dem Server ausgeführt wird. Das Verhindern von jeglichem bösartigem Code ist keine einfache Aufgabe, da dabei gleichzeitig kein valider Lösungsansatz der SuS gestört werden darf. Ein Filtern des Schülercodes über reguläre Ausdrücke könnte den größten Teil der möglichen Attacken verhindern, es bliebe allerdings ein Restrisiko. Zudem muss der Code nicht bösartig sein, um den Server zu belasten. Endlosschleifen oder ineffiziente Algorithmen, die im Code der SuS zu erwarten sind, müssen erkannt und verlässlich abgebrochen werden können. Zudem ist die Visualisierung und das Stoppen der Algorithmen, insbesondere bei schnellerer Ausführung des Codes, durch die Latenzzeit stark beeinträchtigt. Aufgrund dieser Einschränkungen und Herausforderungen ohne direkt absehbare Lösungsansätze lag der Fokus auf einer clientseitigen Lösung.

Bei der clientseitigen Verarbeitung des Python Codes muss auf die Vorarbeit Anderer aufgebaut werden. Hierfür existieren über 20 Tools, die eine Ausführung von Python im Browser ermöglichen. Zu diesem Zweck wird Python Code automatisch in JavaScript übersetzt, welcher dann ausgeführt werden kann. Die beiden umfassendsten und aktiv unterstützten Tools sind hierfür Skulpt[17] und Brython[19]. Brython bietet im Gegensatz zu Skulpt eine zumindest grundlegende Dokumentation. Bei beiden Tools liegt die Herausforderung bei der verzögerten Ausführung von Code, da in JavaScript kein Äquivalent

zu `time.sleep()` existiert. Die Entwickler von Skulp arbeiteten im Verfassungszeitraum dieser Arbeit aktiv an der Einführung von *Suspensions* die eben dieses Problem adressieren sollen. Allerdings sind diese *Suspensions* noch nicht über die Testphase hinaus entwickelt[18]. Daher wurde Brython mit den Möglichkeiten des `browser.timer` Moduls und der Funktion `request_animation_frame()` näher untersucht. Nach verschiedenen Tests mit beiden Funktionalitäten von Brython musste festgestellt werden, dass lediglich wiederholte Ausführung von argumentlosen Funktionen möglich ist, nicht jedoch das zwischenzeitliche Aussetzen des Codes. Diese Funktionalität würde ausreichen, um die korrekte Ausführung der Algorithmen aus der IMP zu visualisieren. Da allerdings auch vorher unbekannter Code der SuS visualisiert werden können muss, wäre eine Anwendung von Brython enorm komplex. Die Struktur des Codes, der tatsächlich ausgeführt wird muss auf einer Funktion aufgebaut sein, die bei wiederholtem Aufruf ohne Argumente immer den nächsten Schritt eines Algorithmus ausführt. Dies wäre über globale Variablen möglich. Gleichzeitig ist aber das automatische Übersetzen des unbekannten, anders strukturierten Schülercodes eine enorm komplexe und fehleranfällige Aufgabe.
Neben diesem entscheidenden Nachteil bei einer clientseitigen Weblösung ist zu erwähnen, dass das Exception Handling bei der Nutzung von Brython anders verläuft als bei Python. Daher können Fehler der SuS nicht abgefangen und in leichter zu verstehender Form ausgegeben werden. Insgesamt ist somit eine Umsetzung im Browser, ob auf Client- oder Serverseite, die den im Kapitel 2 aufgeführten Anforderungen genügt nicht möglich.

4.2.2 GUI Framework

Da für die Benutzeroberfläche keine Weblösung in Frage kommt, muss ein eigenes GUI erstellt werden. Dafür ist zunächst eine Auswahl aus dem großen Angebot an GUI Frameworks zu treffen. Ein Faktor hierbei ist die Lizenz, die eine nicht kommerzielle Nutzung des Frameworks erlauben muss. Das Angebot an solchen Frameworks ist enorm[20]. Es liegen diesem großen Angebot allerdings fast ausschließlich die plattformübergreifenden Technologien Gtk, Qt, Tk und wxWidgets zugrunde. Ein erschöpfender Vergleich des gesamten Angebots bezüglich der Eignung für eine Programmierumgebung ist im Rahmen dieser Arbeit nicht möglich. Nach den Dokumentationen der beiden meist verbreiteten Frameworks für Python Anwendungen Tk und Qt eignen sich scheinbar beide für eine Programmierumgebung mit den oben aufgeführten Anforderungen[22, 21]. Allerdings bietet PyQt mit dem Qt Framework einige Vorteile:

- Das Signal Slot System mit einem Event Loop erlaubt das gleichzeitige Bedienen der GUI Elemente und Ausführen von Code.
- Es existieren verschiedene Methoden nicht nur zur Abbildung, sondern auch zur Animation von Grafikobjekten (`QPropertyAnimation` und `QGraphicsItemAnimation`).

- Es existieren hilfreiche vorprogrammierte Widgets wie z.B. Slider, Textfelder, Colorpicker oder Widgets zum Öffnen und Speichern von Dateien.

Der große Nachteil ist die Tatsache, dass PyQt, nicht wie das mit Python mitgelieferte Tkinter, installiert werden muss.
Die Frage, welches dieser beiden Frameworks besser geeignet ist lässt sich nur durch eine vollständige Umsetzung des Projektes in beiden erschöpfend beantworten. Aufgrund der Zeitbeschränkung, den aufgeführten Vorteilen von PyQt und der Präferenz des Autors nach einigen Tests fiel die Wahl daher auf PyQt.

4.2.3 GUI Aufbau

Ein `QMainWindow` Objekt kann eine Menüleiste, einen Hauptbereich und eine Statusleiste abbilden, in der Informationen zu den Widgets unter dem Mauszeiger angezeigt werden. Im Hauptbereich bietet PyQt ein Layout System in dem horizontale oder vertikale Layoutboxen automatisch an den verfügbaren Platz angepasst werden. Der Layoutbox werden dann Widgets wie Buttons und Textfelder, aber auch weitere Layoutboxen zugewiesen.
Für eine didaktisch sinnvolle Programmierumgebung ist das Ziel die Eingabe, Ausgabe, Anleitung und Visualisierung gleichzeitig im Blick zu haben. Eine Verteilung auf mehrere Fenster, wie z.B. Editor und Konsole, wurde daher ausgeschlossen. Da die Programmierumgebung für spezielle, vom Umfang her sehr begrenzte Aufgaben gedacht ist, kann die Größe des Eingabebereichs für Code im Vergleich zu anderen Editoren vermindert werden. Außerdem sollte die Einblendung der Aufgabenbeschreibung, die mit eventuellen ergänzenden Beispielen recht groß ausfallen kann, optional gestaltet sein.
Da die Visualisierung unterschiedlicher Module völlig unterschiedlich ausfallen kann ist es schwierig, sich von vornherein auf ein finales Layout festzulegen. Grundsätzlich ist eine Umstellung des Layouts durch das Modul möglich. Somit könnten später Algorithmen, deren Visualisierungen einen höheren Platzbedarf haben, das Layout nach Bedarf umstellen.

4.2.4 Aufgabenbeschreibung

Die Aufgabenbeschreibung hat mehrere Zwecke. Sie soll nie für sich alleine stehen, sondern vielmehr bereits im Unterricht behandelte Konzepte wieder aufgreifen und evtl. rekapitulieren. Sie soll das Ziel des zu schreibenden Programmes deutlich machen und zu diesem Zweck die zur Verfügung gestellten Objekte vorstellen. Das könnte für einen Sortieralgorithmus eine zufällig sortierte Liste und für einen Graphenalgorithmus ein vorgefertigter, zufälliger Graph sein. Es ist hierbei wichtig, dass die gegebenen Objekte tatsächlich einen Zufallsaspekt beinhalten, einerseits um den Algorithmus mit verschiedenen Fällen zu testen, andererseits um zu verhindern, dass die SuS das Ergebnis manuell eingeben statt algorithmisch zu lösen.

Um eine ansprechende Präsentation der Aufgabe zu ermöglichen und eventuelle Abbildungen einbauen zu können, bietet sich die Möglichkeit an in PyQt mit `QTextEdit` Feldern HTML Code darzustellen. Die entsprechende HTML Datei wird bei der Auswahl eines Moduls eingelesen und dargestellt.

4.2.5 Code Editor

Der Code Editor ist das Zentrale GUI Element. Er basiert auf einem `QPlainTextEdit` Widget, das um Zeilennummern und einen `QSyntaxHighlighter` ergänzt wurden. Dem `QSyntaxHighlighter` werden die Zielformate verschiedener Syntax, wie z.B. Befehlen, Funktionsdefinitionen, Strings und Zahlen vorgegeben und mit regulären Ausdrücken verknüpft, die die entsprechende Syntax beschreiben. Außerdem wird während der Ausführung des Codes direkt im Editor durch das grüne Hinterlegen der Zeilen hervorgehoben, welche Zeile gerade ausgeführt wird. Tritt ein Error auf, wird die betroffene Zeile rot hinterlegt. Da speziell die Eingabe von Python Code erleichtert werden soll, wird bei Betätigung der Tabulator Taste kein Tab eingefügt, sondern durch das Abfangen des Signals mit der `keyPressEvent` Methode mit 4 Leerzeichen ersetzt.

4.2.6 Ausführung des Schülercodes

Die Ausführung des Schülercodes ist auf verschiedene Arten möglich. Zahlreiche Aspekte der Programmierumgebung hängen davon ab, auf welche Art der Schülercode ausgeführt wird. So muss z.B. eine Bedienung der Benutzeroberfläche während der Ausführung möglich sein, gleichzeitig muss der Schülercode Einfluss auf die Konsole und den Animationsbereich haben. Die Ausführung ist daher durch einen extra Thread, den Import des Schülercodes oder den `exec()` Befehl möglich.

Da PyQt verwendet wird, muss bei der Einführung von Multithreading auf `QThread` zurückgegriffen werden. Bei Verwendung von diesen Threads muss jedes `QObject` einem `QThread` zugewiesen sein. Dies ist problematisch, da der GUI Code auf das `QGraphicsView` Widget, das die Animationen zeigt, zugreifen können muss. Gleichzeitig muss aber auch der Modulcode bzw. als Konsequenz auch der Schülercode auf das selbe Objekt zugreifen können. Dem Autor war es nicht möglich, diese Einschränkung zu umgehen.

Der Import des Modul- oder des Schülercodes ist ebenfalls problematisch, da später auch die Möglichkeit bestehen soll dem Autor unbekannte Module ohne Veränderung des GUI Codes zu importieren. Das Aufrufen von noch nicht existierenden Funktionen in unbekannten Modulen verursachte zahlreiche Probleme. Zudem wird der Code nun direkt in dem GUI Code ausgeführt. Dies macht das Abbrechen von Endlosschleifen ohne das GUI zu schließen schwierig.

Über die `exec(string[, globals[, locals]])` Funktion in Python kann ein String als Python Code ausgeführt werden. Dieser Befehl stellt, wie die anderen Methoden auch, ein Sicherheitsrisiko dar, da die SuS auch schädlichen Code ausführen können. Allerdings kann der Code vor dem Ausführen als String manipuliert werden. Der Vorteil an der Codeausführung über `exec()` liegt darin, dass der ausgeführte Code über einen `raise Error` Befehl abgebrochen werden kann, ohne das GUI zu schließen. Damit nun der ausgeführte Code auf Elemente des GUIs zugreifen kann muss das Window Objekt übergeben werden. Dies wird möglich gemacht, indem das Dictionary der globalen Variablen `globals()` um das Window ergänzt und erst dann an den `exec()` Namespace weitergegeben wird.
Da die SuS nicht selbst für die Visualisierung verantwortlich sein sollen, wird der Modulcode benötigt. Dieser wird als Wrapper bezeichnet immer vor und nach den Schülercode konkateniert, bevor dieser ergänzte Schülercode ausgeführt wird. Je nach Modul wird die Visualisierung dann auf unterschiedliche Art und Weise implementiert.
Unabhängig vom Modul werden Änderungen am Schülercode vorgenommen. Vor der Ausführung wird immer vor und nach den Code der modulspezifische Pre- und Postwrapper konkateniert. Um während der Ausführung des Codes hervorzuheben, welche Zeile gerade ausgeführt wird, den Sprung zur nächsten Zeile zu verzögern und um eine Möglichkeit zu haben die Programmschritte zu zählen, wird nach jeder Zeile eine Funktion `nextline()` eingefügt. Dabei ist auf IndentationError zu achten, die bei falschem Einrücken auftreten können. Ein großes Problem stellt sich nun aufgrund von Python Strukturen wie `if...elif...else`. Vor einem `elif` Statement kann nicht auf dem selben Einrückungsgrad `nextline()` stehen, da ansonsten eine `elif` ohne vorangehendes `if` zu Syntaxfehlern führt. Diese Herausforderung konnte nicht komplett bewältigt werden. Stattdessen werden vor `else` und `elif` Statements keine `nextline()` eingefügt. Bei `else` Statements ist das nicht weiter problematisch, es finden in Zeilen mit else keine interessanten Berechnungen statt. Bei `elif` Statements wird die Programmausführung auch ohne `nextline()` verzögert, sofern ein durch das Modul vorgegebenes Objekt aufgerufen wird. Syntaxfehler werden nach wie vor mit der korrekten Zeilennummer erkannt. Lediglich die Tatsache, dass während der Ausführung der `elif` Zeile im Schülercode die falsche `if...` Zeile markiert bleibt ist zu bemängeln.

4.2.7 Verzögerte Programmausführung

Um den SuS die Möglichkeit zu geben dem Geschehen zu folgen wurde ein Slider implementiert, an dem sich die Verzögerungszeit zwischen den einzelnen Programmschritten zwischen null und zwei Sekunden auswählen lässt. Dabei ist zu erwähnen, dass die genaue Bedeutung dieser Schrittdauer ausschließlich vom Modulcode abhängt. Das bedeutet, dass nicht zwangsläufig jede Zeile Code in der Schrittdauer ausgeführt wird. Vielmehr wird im Modul z.B. der Zugriff auf ein Listenelement registriert, welcher dann eine Animation

und eine Unterbrechung des Programms für die Schrittdauer auslöst. Somit dauert der Vergleich von zwei Listenelementen zwei mal die Schrittdauer. Damit auch Zeilen, in denen kein vorgegebenes Objekt aufgerufen wird verlangsamt ausgeführt werden ist eine Verzögerung in der `nextline()` implementiert.

Um den GUI Code nicht zu stören wird nicht die einfachste Umsetzung mit `time.sleep(Schrittdauer)` gewählt, die das GUI einfriert. Stattdessen wird ein `QTimer` und ein neuer `QEventLoop` gestartet, in diesem neuen `QEventLoop` können GUI Operationen registriert und Animationen ausgeführt werden. Nach dem Ablauf des `QTimers` wird der neue `QEventLoop` beendet und der Schülercode wird an entsprechender Stelle weiter ausgeführt.
Außerdem haben die SuS die Möglichkeit, die Programmausführung durch einen Pause Button zu unterbrechen und mit einem Step Button schrittweise auszuführen. Hierfür wird im Modulcode kein Timer gestartet, sondern der `QEventLoop` nur durch den Step Button geschlossen. Die Implementierung eines Zurück Buttons ist nur mit großem Aufwand möglich. Es wäre notwendig eine Historie aufzubauen, in der für jeden Schritt die aktuelle Codezeile, Konsolenausgabe und der Zustand jeder einzelnen Variable und jedes Objektes gespeichert wird. Würde eine solche Historie implementiert, wäre außerdem eine Unterbrechung des Codes nicht mehr notwendig. Es wäre ein Durchwandern der Historie in beide Richtungen möglich. Rückblickend ist dieser Ansatz interessant, auch wenn er andere, neue Herausforderungen mit sich brächte. Der Zeitbedarf für das Umschreiben der bis dato verfassten Programmierumgebung inklusive der Module ist für den verbliebenen Bearbeitungszeitraum zu groß.

4.2.8 Fortschrittsbalken

Um den SuS einen Überblick über den Fortschritt ihres Programmes zu bieten, wurde ein Fortschrittsbalken implementiert. Dieser zeigt den prozentualen und absoluten Anteil der bereits abgearbeiteten Programmschritte an. Da die genaue Anzahl der Schritte bei einem unbekannten Algorithmus auf einem zufällig generierten Objekt im Voraus nicht bekannt ist, wird der Schülercode bei jedem Ausführen tatsächlich zweimal ausgeführt. Einmal ohne Visualisierung und Konsolenausgabe und mit einer Schrittdauer von null Sekunden, wobei über die `nextline()` Funktion die Schritte gezählt werden, und dann erneut für die SuS sichtbar. Beim ersten Durchlaufen werden Errormeldungen ignoriert, es wird lediglich darauf hingewiesen, dass ein Error existiert. Daraufhin wird der Schülercode bis an die Stelle ausgeführt, an der der Fehler auftritt. Die Abhandlung der Fehler ist im nächsten Kapitel 4.2.9 beschrieben.
Vor dem ersten Ausführen wird ein Timer gestartet. Benötigt das Programm ohne künstliche Verlangsamung und ohne Konsolenausgabe länger als zwei Sekunden, so werden die SuS über eine mögliche Endlosschleife im Code informiert und es wird die Möglichkeit des

manuellen Beendens des Programms empfohlen.

4.2.9 Fehlermeldungen

Um den SuS leicht verständliche Hilfe bei üblichen Fehlern zu bieten, werden Errormeldungen, die beim Ausführen des Schülercodes auftreten über `try... except` abgefangen. Da der Schülercode für den Fortschrittsbalken zweimal ausgeführt wird und bereits beim ersten Mal abbricht, wird der Schüler nach der ersten, quasi sofortigen Ausführung darüber informiert, dass sein Code einen Fehler enthält. Damit der Fehler besser nachvollziehbar ist, wird der Code im Anschluss (aber ohne die nun nicht mögliche Fortschrittsanzeige) ausgeführt bis der Fehler auftritt. Die SuS erhalten nun eine fehlerspezifische Fehlermeldung auf deutsch mit Hinweisen, wie der Fehler meist behoben werden kann. Um zu erkennen, welche Art von `Exception` vorliegt und welche Codezeile ihn hervorruft, können zwei Methoden verwendet werden: Die scheinbar einfachste Methode ist das Zugreifen auf die der `Exception` übergebenen Argumente `Exception.args`, wie z.B. im Falle des `IndentationError`. Allerdings sind nicht bei allen Arten von `Exception` alle relevanten Informationen enthalten, wie z.B. bei dem `ZeroDivisionError`. Daher muss die andere Methode gewählt werden, um die Details der `Exception` zu verarbeiten: Mit dem Befehl `sys.exc_info()` können ebenfalls `Exception` ausgelesen werden: `exc_type, exc_value, exc_traceback = sys.exc_info()`. Die unterschiedlichen Ergebnisse bei den verschiedenen Arten von `Exception` werden in Tabelle 2 beispielhaft gegenübergestellt. Es wird in der Tabelle deutlich, dass die Zeilennummer und der String mit dem fehlerbehafteten Code zwar in den Argumenten des `IndentationError` zu finden ist, allerdings nicht in den Traceback Informationen, wobei beim `ZeroDivisionError` zumindest die korrekte Zeilennummer vom Traceback ausgegeben wird, jedoch in anders formatierter Form. Es ist demnach erforderlich, jede Art von `Exception` getrennt zu betrachten um die relevanten Informationen zu extrahieren.
Es werden folgende Arten von `Exception` abgefangen:

- `Infanteriedivision`
- `NameError`
- `IndexError`
- `IndentationError`
- `KeyError`
- `TypeError`
- `SyntaxError`
- `AttributeError`
- `UnboundLocalError`

Da der ausgeführte Code nicht mit dem Code übereinstimmt, den die SuS schreiben und sehen, sind die Zeilennummern für sie nicht aussagekräftig. Während es noch möglich wäre die Zeilenanzahl des pre_wrappers von der Zeilennummer zu subtrahieren, machen

Objekt	IndentationError	ZeroDivisionError
Exception.args	('unindent does not match any outer indentation level', ('<string>', 75, 29, ' for j in range(len(liste)):\n'))	('division by zero',)
exc_type	<class 'IndentationError'>	<class 'ZeroDivisionError'>
exc_value	unindent does not match any outer indentation level (<string>, line 75)	division by zero
traceback.format_tb (exc_traceback)	[' File pyqt_main.py", line 284, in run_code\n exec(code, passed_globals)\n']	[' File pyqt_main.py", line 284, in run_code\n exec(code, passed_globals)\n', ' File "<string>", line 75, in <module>\n']

Tabelle 2: Gegenüberstellung des Verhaltens verschiedener Exceptions

die bei manchen Modulen notwendigen Einschübe innerhalb des Codes eine direkte Verwendung der im Traceback gegebenen Zeilennummer unmöglich. Stattdessen wird die Zeilennummer über die von `nextline()` vorgegebene Zeile bestimmt. Im Fall des Syntax Fehlers ist dies nicht möglich, da er nicht zur Laufzeit auftritt. Daher wird entweder direkt der fehlerhafte String, oder der in der entsprechenden Zeile des ergänzten Codes stehende String verwendet, um den für SuS sichtbaren Code zu durchsuchen, so kann die für SuS nutzbare Zeilennummer ermittelt werden. Natürlich kann es Situationen geben, in denen der String in mehreren Zeilen vorkommt. Bei den gegebenen Aufgaben ist dies aber nicht zu erwarten. Außerdem wäre eine Zeile mit Syntaxfehler mit hoher Wahrscheinlichkeit an allen Stellen falsch, an der sie vorkommt. Daher wird diese improvisierte Lösung akzeptiert.

4.2.10 Konsolenausgabe

Die `print()` Befehle der SuS müssen in dem Konsolen `QPlainTextEdit` Feld dargestellt werden. Zu diesem Zweck wird im Modulcode der `print()` Befehl überschrieben und die Argumente an die `newprint()` Funktion übergeben. Diese `newprint()` Funktion führt `__builtins__.print()` aus, gibt also die Eingabe auch an die tatsächliche Konsole aus. Zudem wird mithilfe des `io.StringIO` Moduls `sys.stdout` kurzzeitig überschrieben, der Datenstream von `print()` abgefangen und als Text an die Konsole im GUI übergeben. Somit muss die `print()` Funktion nicht komplett neu implementiert werden.

4.2.11 Visualisierung

Für die Visualisierung wird ein `QGraphicsView` Widget verwendet, das eine `QGraphicsScene` darstellt, die ausschließlich von dem Modulcode mit `QGraphicsItem` und `QGraphicsObject` bevölkert wird. Im Detail wird das Vorgehen daher in den entsprechenden Kapiteln der Module diskutiert.

Insgesamt wird bei der Visualisierung weniger Wert auf schöne Grafiken, dafür viel Wert auf Animationen gelegt. Grafiken können die Programmierumgebung zwar für SuS ansprechender gestalten, sind aber der Übersicht und dem Verständnis nicht zwingend zuträglich, bei zu ausgefallenem Stil sogar abträglich. Zudem sind sie in der Gestaltung zeitaufwändig. Animationen sind zwar ebenfalls aufwändig, richtig angewendet machen sie das Verständnis der betrachteten Algorithmen jedoch deutlich einfacher. So wird z.B. in dem Bubblesort Modul eine Ziffer in einem Listenelement nicht von einem Frame auf den nächsten überschrieben. Dies kann den SuS leicht komplett entgehen. Selbst wenn die Änderung als solche registriert wird, ist die ursprüngliche Ziffer unbekannt. Dieses Problem wurde bereits beim Testen von Kara deutlich (vgl. Kapitel 8). Mit einer Animation wie z.B. dem Verblassen einer Ziffer und dem Erscheinen einer neuen wird der Fokus auf das gerade relevante Element gezogen und die SuS haben genug Zeit zu erkennen, was der Ist-Zustand ist, bevor die Änderung in Kraft tritt. Aus diesem Grund ändert sich auch in Graphen Modulen die Farbe eines Knotens nicht schlagartig, sondern blinkt vor der Änderung einige Male.
Für die Animationen wurden `QPropertyAnimation` und `QGraphicsItemAnimation` verwendet. `QGraphicsItemAnimation` wird im Zusammenspiel mit einer `QTimeline` verwendet, um zu bestimmten Zeitpunkten in der Timeline Eigenschaft eines `QGraphicsitem` festzulegen. Die Zwischenschritte werden dann automatisch interpoliert.
`QPropertyAnimation` bietet zusätzlich die Möglichkeit, Eigenschaften von allen Objekten, die von `QObject` erben zu animieren. `QPropertyAnimation` benötigt keine `QTimeline`, stattdessen genügt die Angabe eines Start- und Endwertes einer Objekteigenschaft die animiert werden soll, sowie der Animationsdauer. Da `QGraphicsItem` nicht von `QObject` erbt, wird stattdessen das für solche Zwecke zur Verfügung gestellte `QGraphicsObject` verwendet. Diesem Objekt können, neben einigen Standardeigenschaften, auch zusätzliche Eigenschaften durch eine `fset` Funktion zugewiesen werden. Es ist also möglich individuell angepasste Animationen für `QGraphicsObject` zu generieren. Daher werden in den Modulen soweit möglich `QPropertyAnimation` verwendet.

4.2.12 Ausgabe der lokalen Variablen

Ähnlich wie im oben vorgestellten Tool Pythontutor sollen die von den SuS eingegebenen lokalen und globalen Variablen angezeigt werden. Da allerdings nicht nur der Schülercode, sondern auch der Modulcode ausgeführt wird, befinden sich zahlreiche für die SuS verweirrenden und unnützen Variablen in dem Namespace. Es werden daher zunächst alle nicht von den SuS deklarierten Variablen gespeichert. Dafür wird über die Funktion `locals()` vor der Ausführung des Schülercodes festgestellt, welche Variablen bereits existieren. Mit jeder Ausführung der `nextline()` Funktion werden nur neue lokale Variablen im entsprechenden Bereich im GUI dargestellt. Allerdings musste festgestellt werden, dass bei mehrfachem Ausführen der `exec()` Funktion der Namespace nicht zurückgesetzt wird.

Dies gilt auch, obwohl `exec()` explizit der globale Namespace übergeben wird (vgl. Kapitel 4.2.6). Dies führt dazu, dass zu Beginn der für die SuS sichtbaren zweiten Ausführung des Codes einzelne Variablen noch auf dem Endzustand des vorherigen Durchlaufs stehen. Da im ersten Durchlauf aber nur dann Variablen in den Namespace eintreten, wenn sie neu definiert werden, ist ein Überschreiben der alten Werte im nächsten Durchgang garantiert. Die Funktionalität des Programmes ändert sich also nicht.

4.2.13 Ausführbare Datei

Es ist davon auszugehen, dass an Computerräumen in Schulen, sowie auf den privaten Geräten der SuS keine Python Umgebung installiert ist. Daher wurde mithilfe der Entwicklungsversion von Pyinstaller 3.3.1[33] eine unter Windows ausführbare .exe Datei erstellt. Um dies zu erreichen, musste das gesamte Projekt von PyQt4 auf PyQt5 umgezogen werden, was wiederum die Verwendung von Python 3.5 notwendig machte. Während für PyQt5 nur die C++ Dokumentation zur Verfügung steht, decken sich die meisten Befehle mit der Vorgängerversion. Um die generierte Datei ausführen zu können, mussten zudem manuell fehlende .dll Dateien eingefügt werden. Eine Anleitung dazu findet sich in der Readme der Programmierumgebung.

4.2.14 Module Bubblesort und Insertionsort

Der Bubblesort Algorithmus ist einer der anschaulichsten Sortieralgorithmen. Obwohl er mit der Laufzeit $O(n^2)$ nicht schnell ist, vermittelt er ein wichtiges Konzept für die SuS: Der PC hat keinen Überblick über die gesamte Liste wie ein Mensch ihn hat. Um dieses Konzept noch weiter zu veranschaulichen, wird die Liste als eine Reihe von Schubladen visualisiert. Dabei ist zunächst nur die Nummer der Schublade (der Index) zu sehen, erst wenn dieser Index aufgerufen wird öffnet sich die Schublade und das eigentliche Element, in diesem Modul immer ein Integer, wird sichtbar.
Die Liste mit der die SuS arbeiten sollen wird als `liste` vorgegeben. Sie ist mit zehn Zufallszahlen zwischen 0 und 9 gefüllt. Da der Schülercode, und damit der Modulcode, immer zweimal ausgeführt wird, muss sichergestellt sein, dass beide Male dieselbe Liste bearbeitet wird, daher wird die Liste in der dafür im Hauptprogramm reservierten `moduletemp` Variable gespeichert. Für jedes Listenelement wird dann eine Schublade, ein Schubladenkasten und ein Element als `QGraphicsObject` generiert. Um zu registrieren, welche Listenelemente aufgerufen werden, wurde eine neue Klasse `CustomList` erstellt, die von der in Python integrierten `__builtins__.list` erbt. In der `CustomList` sind die Descriptor `__setitem__` und `__getitem__` überschrieben. Es wird zwar dasselbe Ergebnis wie von `list` zurückgegeben, aber gleichzeitig ein `QEventLoop` gestartet und die Animation der entsprechenden `QGraphicsObject` ausgelöst. Somit können die SuS mit der Liste umgehen, als handle es sich um eine normale Python Liste und sehen den Effekt ihres

Programmes im Animationsbereich. Bei einem Vergleich oder einer Zuweisung, bei der zwei Listenelemente beteiligt sind, also `__getitem__` oder `__setitem__` zweimal in einer Zeile aufgerufen wird, sollen beide Schubladen gleichzeitig geöffnet sein. Daher werden die Schubladen nicht direkt nach dem Öffnen wieder geschlossen. Stattdessen wird mit jeder Ausführung von `nextline` jede Schublade geschlossen.

4.2.15 Modul Listen

Anders als im Bubblesort und Insertionsort Modul müssen beim Listen Modul während der Programmausführung `QGraphicsObject` generiert und entfernt, statt nur animiert werden. Die Generation der dieser Objekte erfolgt daher nicht mehr separat, sondern innerhalb der `CustomList` Klasse. Dafür werden neben den `__setitem__` und `__getitem__`- auch die `append`, `__delitem__` und `__init__` Methoden überschrieben. Um nicht beim ersten Durchgang des immer zwei mal ausgeführten Codes ein `QGraphicsObject` zu löschen, das im zweiten benötigt wird, wird nur die klassische Liste in `moduletemp` zwischengespeichert und die `CustomList` in jedem Durchgang von dieser Liste neu erstellt.

4.2.16 Modul Prüfziffern

Da in diesem Modul keine Visualisierung sinnvoll erschien war die Umsetzung nicht aufwändig. Aus einer Liste von zehn korrekten ISBN Nummern wird eine zufällige Anzahl an zufälliger Stelle manipuliert. Somit ist sichergestellt, dass keine statische Lösung, die immer dieselben Nummern aussortiert, Erfolg haben kann. Die Schwierigkeit für die SuS liegt bei dem Wechsel zwischen Strings und Integern, sowie bei dem Umgang mit Indexen und Schleifen.

4.2.17 Modul Turtle

Im Turtle Modul werden alle Animationen direkt von den SuS ausgelöst, was die Gestaltung des Codes sehr vereinfacht. Das Turtle Objekt beinhaltet eine Liste von Tupeln der zu zeichnenden `QLineF` Objekte, sowie ihrer Farbe und Linienbreite. Die `geradeaus()` Methode hängt ein `QLineF` Objekt mit den aktuellen Optionen für Farbe und Linienbreite an diese Liste an. Sämtliche anderen Funktionen beeinflussen lediglich diese Optionen. Da in PyQt keine vorgefertigte Dreieck Klasse existiert, wird `QPolygonF()` verwendet. Für die Erstellung dieses Polygons muss jeder Eckpunkt einzeln festgelegt werden, was über trigonometrische Formeln mithilfe der aktuellen Orientierung des Turtle Objekts als Winkelangabe zwischen 0°und 360°erfolgt.

4.2.18 Graphenmodule

Im Graphenmodul wird zunächst ein Graph erstellt, auf dem der Schülercode arbeiten soll. Hierbei wird die Anzahl der Knoten, maximale Anzahl der Kanten pro Knoten und

ein Wert zwischen 0 und 1 übergeben. Dieser Wert bestimmt, wie viele der potentiell möglichen Kanten tatsächlich generiert werden. Der resultierende Graph ist also bei einem hohen Wert stark verknüpft, während bei einem Wert von 0 keine Kante existiert. Somit kann der Graph an die gewünschte Aufgabe angepasst werden. Die Verteilung der Knoten ist dabei bei jedem Durchgang gleich, die Kanten sind allerdings unter Beachtung der obigen Einschränkungen zufällig verteilt. Damit der Graph im zweiten Durchlauf des Schülercodes dieselbe Struktur hat wie im ersten Durchlauf, wird vor der Erstellung des ersten Graphen ein zufälliger `random.seed()` festgelegt, der bei der Erstellung des zweiten Graphen wieder aufgerufen wird. Da nun deterministisch exakt dieselben Kanten generiert werden, kann zwischen den Durchläufen die `QGraphicsScene` zurückgesetzt werden und die `QGraphicsObjects` des ersten Graphen müssen nicht zwischengespeichert werden. Um Code auszuführen, sobald die Eigenschaft `markiert` eines Knotens aufgerufen oder verändert wird, wurde `markiert` als `property` mit entsprechendem getter und setter Methoden festgelegt. Innerhalb dieser Methoden kann nun das farbige Blinken des betroffenen Knotens ausgelöst werden. Es wurde dabei ein Blinken der einfachen Farbtransition oder dem noch einfacher umzusetzenden Umspringen der Farbe vorgezogen. Somit können die SuS eindeutiger erkennen, an welchem Knoten gerade welche Veränderung ausgelöst wurde.
Da die `Graph` und `Knoten` Klasse nicht von anderen eingebauten Klassen erbt und damit über `print(Knoten)` keine schönen Ausgaben liefert, wurde die `__str__` Methode überschrieben. Außerdem wird bei der Darstellung der lokalen Variablen die `Knoten` Klasse abgefangen und nur die ID des entsprechenden Knoten dargestellt. Mit der Erstellung zufälliger Start- und Zielknoten können nun von den Schüleralgorithmen Pfade und Erreichbarkeit durch Breiten- und Tiefensuche ermittelt werden. Die Umsetzbarkeit dieser Algorithmen auf einem gerichteten Graphen wird bisher nur durch ein einzelnes Modul demonstriert.

4.2.19 Tests

Zum Testen des Codes einer Anwendung mit Benutzeroberfläche stellt PyQt die `QTest` Klasse zur Verfügung. Im Zusammenspiel mit `unittest` können damit User Interaktionen wie das Klicken auf Buttons simuliert werden. Neben einigen trivialen Operationen wie z.B. dem Wechsel zwischen Vollbild- und Fenstermodus werden alle Module auf verschiedene Inputs getestet. Es wird mit neun absichtlich fehlerhaften Programmen getestet, ob die Errorerkennung den richtigen Fehlertyp und die korrekte fehlerhafte Zeile ausgibt. Für jedes Modul wird außerdem getestet, ob häufig vorkommende Variablennamen, die möglicherweise von SuS im Schülercode verwendet werden, bereits im Modulcode belegt werden. Dies hätte zur Folge, dass sie während der Ausführung nicht als lokale Variable ausgegeben werden und damit für die SuS nicht sichtbar sind. Zudem wird für jedes Modul ein Beispielcode getestet, der die jeweilige Aufgabe löst. Einschränkungen des Test-

verfahrens ergeben sich durch die Tatsache, dass während der Programmausführung ein neuer `QEventLoop` gestartet wird und somit Interaktionen mit dem GUI über `QTest` nicht möglich sind. Es kann also z.B. das Verhalten der Start und Stop Buttons im Steuerbereich nicht getestet werden. Ebensowenig kann die vom Modulcode vorgegebene Animation getestet werden. Es kann allerdings wie oben beschrieben getestet werden, ob der Schüler- und Modulcode als Ganzes ohne Fehlermeldung abschließt.

5 Analyse und Evaluation der Programmierumgebung

5.1 Anforderungen

Es wird nachfolgend der Erfüllungsgrad des in Kapitel 2 aufgeführten Anforderungskatalogs geprüft.

Programmiersprache - erfüllt

Mit Ausnahme des Turtle Moduls sind sämtliche von SuS programmierten Funktionen, Methoden und Datenstrukturen nativ in Python. Der Schülercode kann mit Zusatz der von den Modulen zur Verfügung gestellten Objekten auch außerhalb der Programmierumgebung ausgeführt werden.

Sprache - erfüllt

Die Anleitung und Ausgabe sowie das GUI sind in deutscher, für SuS geeigneter Sprache verfasst. Lediglich nicht abgefangene Fehlermeldungen werden auf Englisch angezeigt

Abdeckung der IMP Inhalte - teilweise erfüllt

Die IMP Inhalte werden nicht komplett abgedeckt. Die Abdeckung der Themen, für die das Programmieren der SuS sinnvoll wäre, ist aber durch weitere Module möglich.

Vorkenntnis - erfüllt

Die Module sind darauf ausgerichtet nach einer Unterrichtseinheit über das Thema absolviert zu werden. Sämtliche dafür notwendige Informationen werden in der Anleitung gegeben. Es ist Vorkenntnis über Python Syntax von Vorteil, allerdings sind Module zur Vermittlung der Syntax leicht zu implementieren, ein Beispiel ist das *Listen* Modul. Die Module *Freies Programmieren* und *Turtle* können ebenfalls unter Anleitung für die Vermittlung der Syntax verwendet werden.

Spezielle Syntax - erfüllt

Es sind keine nicht nativ in Python zur Verfügung stehenden Befehle notwendig.

Themenfokus - erfüllt

Jedes Modul kann speziell um einen Algorithmus oder Thema herum aufgebaut sein und nur dieses visualisieren. Es ist trotzdem auch das Verfassen anderer Programme möglich.

Visualisierung - teilweise erfüllt

Das GUI ist für Schüler nicht sehr Ansprechend gestaltet. Die Visualisierung ist funktional und durch Animationen übersichtlich gestaltet. Nicht alle Module enthalten Visualisierungen.

Verlangsamte Ausführung - erfüllt

Der Schülercode kann in beliebiger Geschwindigkeit abgespielt werden. Außerdem kann die Ausführung gestoppt und schrittweise fortgesetzt werden. Es ist jedoch kein Rückwärtsschritt möglich.

Hilfestellung und Umgang mit Fehleingaben - teilweise erfüllt

Die Anleitung bietet eine statische Hilfestellung, während Fehlermeldungen mit Lösungsansätzen präsentiert werden. Der Code kann bis zu dem Auftreten der Fehlermeldung ausgeführt und visualisiert werden. Bei Syntaxfehlern wird nur auf die Zeile verwiesen, nicht aber auf spezielle Syntaxfehler wie z.B. *Es fehlt ein : nach else.*

Fortschrittssicherung - teilweise erfüllt

Der Schülercode kann jederzeit lokal gespeichert werden. Bei jedem Ausführen und Schließen des Programms, sowie alle fünf Sekunden wird außerdem automatisch gespeichert. Darüber hinaus ist der Code nicht erneut verfügbar. Die Lehrkraft hat keine Möglichkeit, den individuellen Schülerfortschritt zu überprüfen. Eine Userverwaltung und ein Server, dem von der Programmierumgebung der Fortschritt des Users gemeldet wird, wäre eine für den Lehrer hilfreiche Ergänzung.

Verfügbarkeit - erfüllt

Durch die Implementierung als ohne Installation ausführbare .exe Datei ist die Verfügbarkeit für Schulen sehr gut. Es ist sogar möglich, ohne Python installieren zu müssen neue Module zu programmieren und aufzunehmen.

Datenschutz - erfüllt

Es werden keine Daten der SuS benötigt oder gespeichert.

Sicherheit vor bösartigem Code - teilweise erfüllt

Endlosschleifen werden erkannt und sind unterbrechbar. Es kann allerdings lokal grundsätzlich bösartiger Code ausgeführt werden. Da z.B. für Bildmanipulation auch Module vorstellbar

Kriterium (Kapitel)	Code.org	Pythontutor.com	Codecademy	Turtle	Greenfoot	Kara	Scratch	Bluej	Klassische Programmierumgebung	Eigene Programmierumgebung
2.3 Prog. Sprache	Graf.	Py	Versch.	Py	Java	Py	Graf.	Java	Versch.	Py
2.4 Sprache	D	E	E	D	D	D	D	D	D	D
2.5IMP Inhalte	-	-	-	-	0	-	-	0	0	0
2.7 Spez. Syntax	+	-	-	+	+	+	+	-	-	-
2.8 Themenfokus	-	-	-	-	-	-	-	-	-	+
2.9 Visualisierung	+	0	-	+	+	+	+	-	-	0
2.10 Verlangsamung	+	+	-	+	+	+	-	-	0	+
2.11 Hilfestellung	+	-	+	0	0	+	0	0	0	0
2.12 Fortschrittssich.	+	-	+	+	+	+	+	+	+	0
2.13 Verfügbarkeit	+	+	+	+	+	+	+	+	+	+
2.14 Datenschutz	-	+	-	+	+	+	+	+	+	+
2.15 Sicherheit	+	+	+	-	-	+	+	-	-	0

Tabelle 3: Eigene Programmierumgebung im Vergleich mit Alternativen

sind, die mit anderen Dateien auf dem System interagieren sollen, ist es schwer eine Grenze zu ziehen, welche Befehle im Schülercode grundsätzlich gesperrt sein müssen. Da der Python Code allerdings auf einem PC mit nur eingeschränktem Zugriff auf das Schulnetzwerk ausgeführt wird, besteht durch die Programmierumgebung kein relevant gesteigertes Risiko.

Insgesamt ist es gelungen eine Programmierumgebung zu erstellen, die für die Verwendung im IMP Unterricht besser geeignet ist als die untersuchten Alternativen (vgl. Tabelle 3). Gleichzeitig sind einzelne Anforderungen nicht erschöpfend erfüllt. Einige dieser Schwachstellen werden in Kapitel 6 aufgegriffen.

5.2 Evaluation

Eine kurzfristige empirische Evaluation der erstellten Programmierumgebung gestaltet sich schwierig. Die Programmierumgebung ist für einen langfristigen Einsatz über drei Schuljahre in Verbindung mit dem Informatikunterricht gedacht. Weder steht die Zeit, noch eine Versuchsgruppe für einen ausführlichen Test zur Verfügung. Selbst wenn dem so wäre, existierten keine Vergleichsdaten, da der Informatikunterricht in dieser Form vor dem Schuljahr 2018/19 nicht stattfand. Die Wirksamkeit der Programmierumgebung hängt zusätzlich stark von den vorangehenden Unterrichtseinheiten ab.
Um trotzdem eine qualitative Rückmeldung über die Validität der Programmierumgebung zu erhalten, wurde in Zusammenarbeit mit dem Ganztagsgymnasium Osterburken (GTO) und im Speziellen dem Informatiklehrer Christian Schinnagel eine Benutzerstudie

durchgeführt. Dabei konnte die Programmierumgebung mit dem zweistündigen Informatik Kurs der Oberstufe über drei Schulstunden getestet werden. Die Oberstufe ist nicht exakt die Zielgruppe der Programmierumgebung, allerdings hatten die beteiligten Schüler (es gab keine Schülerinnen im Kurs) weder den Informatik Aufbaukurs der siebten Klasse, der nach ihrer Zeit eingeführt wurde, noch das IMP Profil durchlaufen. Einige der Schüler hatten bereits an einer AG teilgenommen. Der Kenntnisstand entspricht demnach in etwa dem der Zielgruppe, auch wenn das Alter dies nicht tut. Eine jüngere Gruppe von SuS stand für diese Benutzerstudie nicht zur Verfügung.

5.2.1 Ablauf

Der Lehrauftrag der Schule darf durch die Benutzerstudie nicht vernachlässigt werden, daher wurde der Ablauf der Benutzerstudie in Zusammenarbeit mit Herrn Schinnagel geplant. Im optimalen Anwendungsfall der Programmierumgebung sind mehrere Stunden zur Einführung in das Thema notwendig um Fragen zu klären wie: Was ist eine Programmiersprache? Warum wurde Python gewählt? Was ist eine Programmierumgebung? Was ist Syntax? Wie sieht ein Python Programm aus? Welche Befehle werden benötigt werden?
Damit die Schüler möglichst viel Zeit mit der Programmierumgebung verbringen konnten wurden diese Inhalte nach einer kurzen Einführung in der ersten Stunde am 26.09.2018 vom Autor vorgestellt. Dafür wurden über einen Projektor in der Programmierumgebung Codebeispiele von einfachen print() Statements, sowie einer Schleife, die Quadratzahlen bis 100 ausgibt gezeigt. Außerdem wurde das Einverständnis der Schüler für die Auswertung ihrer Daten eingeholt. Diese Einführung dauerte insgesamt 35 Minuten. Den Schülern stand anschließend ein doppelseitiges Handout mit grundlegender Syntax (siehe Anhang IV), sowie die Aufgabenbeschreibungen und Hilfestellungen im Turtle Modul zur Verfügung. Bereits in der ersten Stunde konnte somit bereits selbstständig programmiert werden.
In der zweiten Stunde am 01.10. haben die Schüler bis auf eine etwa fünfminütige Unterbrechung selbstständig gearbeitet. Hintergrund der Unterbrechung war eine kurze Diskussion über zwei verschiedene Lösungswege einer gegebenen Aufgabe, die verdeutlichen sollte, dass verschiedene Lösungswege valide sein können. Außerdem wurde in einer Variante der Lösung die Verwendbarkeit der Schleifenvariable zu anderen Zwecken als nur dem Wiederholungen zählen gezeigt, da dies auch für spätere Aufgaben notwendig war.
In der dritten Stunde am 08.10. wurde die selbstständige Arbeit der Schüler nach den ersten 30 Minuten unterbrochen um einen Evaluationsbogen auszufüllen. Zum Abschluss wurden einzelne Schüler aufgefordert, ihren Code vorzustellen.
Insgesamt ist zu erwähnen, dass alle beteiligten Schüler motiviert und fokussiert gearbeitet haben und keine Ermahnung von Seiten des Lehrers oder des Autors notwendig war. Während der Arbeitsphasen der Schüler haben der Lehrer und der Autor auf Fragen

reagiert und grundsätzlich ermuntert, die Fehlermeldung genau zu lesen, bzw. den entsprechenden Teil des Handouts nochmals anzusehen. Erst falls der Schüler anschließend nicht selbst auf eine Lösung kam wurde weitergeholfen.
Von den acht Schülern, die alle drei Stunden anwesend waren, hatten fünf nach der Gesamtarbeitszeit von 85 Minuten alle Aufgaben im Turtle Modul abgeschlossen. Die beiden in der ersten Stunde nicht anwesenden Schüler waren die Schüler D und I.

5.2.2 Auswertung

Für die Auswertung der Benutzerstudie wurden vier Faktoren herangezogen, die nachfolgend behandelt werden.

Evaluationsbögen
Bei der Evaluation von Tools zur Vermittlung von Informatik ist die Evaluation durch Nutzer die beliebteste Methode [1]. Bei dieser Benutzerstudie soll der Evaluationsbogen drei Hauptzwecke erfüllen:

- Quantitative Erfassung der Vorkenntnisse
- Quantitative Erfassung der Erfahrungen im Umgang mit der Programmierumgebung
- Qualitative Rückmeldung über Mängel und gute Features

Die optionale Angabe des Benutzerkürzels ihres Schullogins nahmen die meisten Schüler aus Annonymitätsgründen nicht wahr, sodass eine Zuordnung zwischen Evaluation und der Auswertung des entsprechenden Schülercodes nicht möglich ist.
Aus der Abbildung 14 geht hervor, dass der Großteil der Schüler keine Erfahrung mit Python hatte. Hier ist zu erwähnen, dass auf die Frage in der ersten Stunde, wer bereits mit Python programmiert hat, im Widerspruch zu den Antworten im Evaluationsbogen kein Schüler reagiert hat. Einige Schüler hatten bereits allgemeine Programmiererfahrung, wobei dies zu Beginn ebenfalls lediglich drei Schüler angegeben hatten. Aus der Grafik geht außerdem folgendes hervor:

- Alle Schüler konnten mindestens teilweise Erfolgserlebnisse machen.
- Ein Schüler wusste mithilfe der Aufgabenbeschreibung nicht, was von ihm verlangt wird.
- Fehlermeldungen und Visualisierungen haben den meisten Schülern dabei geholfen, Fehler im Programm zu finden.
- Die Visualisierung hat allen Schülern dabei geholfen, die zugrundeliegenden Konzepte wie Schleifen und Funktionen zu verstehen.

Abbildung 14: Auswertung des Evaluationsbogens. Es standen die vier Antwortmöglichkeiten trifft (gar nicht/eher nicht/eher/voll) zu zur Verfügung.

- Manche Schüler nutzten die Möglichkeit, Code schrittweise auszuführen.
- Die primäre Hilfestellung zur Fehlerentdeckung hat nicht allen Schülern geholfen, ihre Fehler zu entdecken.

Darüber hinaus wurden die im Anhang I zu findenden qualitativen Antworten gegeben. Dabei ist hervorzuheben, dass sich die Schüler insgesamt positiv über die Umgebung geäußert haben. Die Oberfläche wurde mehrfach als übersichtlich und leicht verständlich bezeichnet, aber auch als visuell wenig ansprechend. Es wurden einige sinnvolle, aber nicht tiefgreifende Verbesserungsvorschläge gemacht wie z.B. ein Undo Button. Kritik wurde auch an den teilweise nicht genau zutreffenden Fehlermeldungen und dem kleinen Animationsbereich geübt.

Auswertung des Schülercodes

Für die Auswertung wurde mit dem Einverständnis der Schüler der Code bei jedem Ausführen gespeichert. Diese Daten können im Auswertungsmodus der Programmierumgebung ausgewertet und betrachtet werden. Neben der Ausgabe einiger im folgenden diskutierten Werte wird eine Grafik wie Abbildung 15 generiert, in der nach Sitzungen getrennt eingesehen werden kann, wann und mit welchem Resultat der Schülercode ausgeführt wurde. Die Grafiken sämtlicher Schüler befinden sich im Anhang VI. Da die Sy-

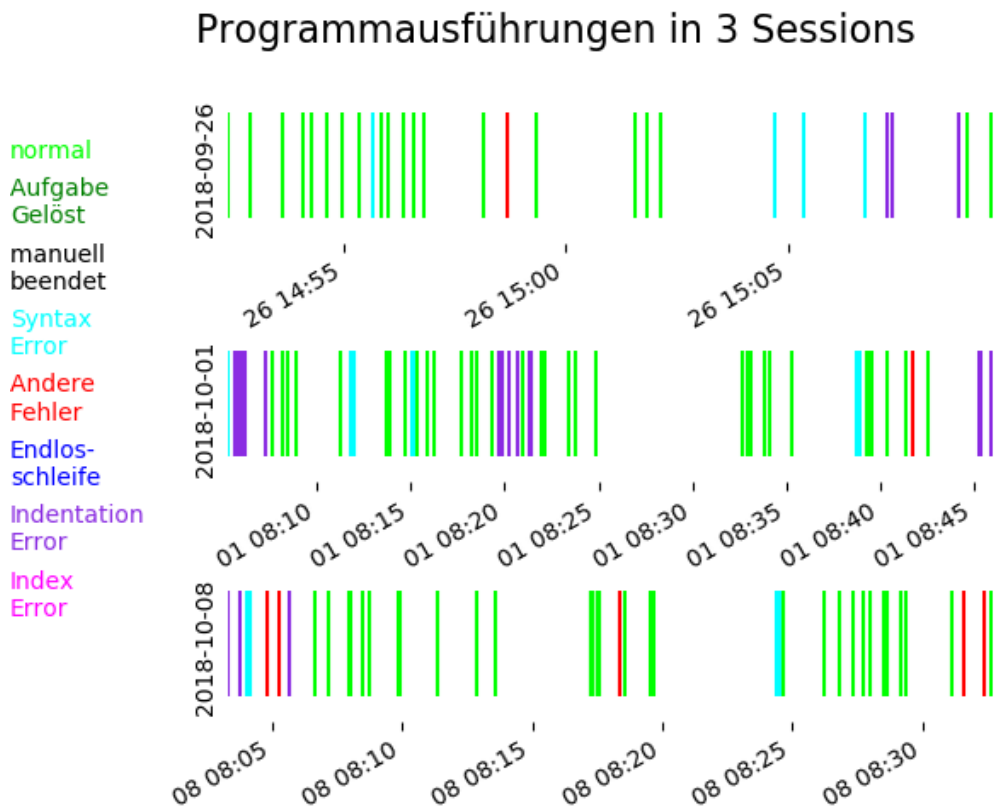


Abbildung 15: Sitzungsprofil des Schülers B.

stemzeit der PCs der Schüler nicht korrekt und an verschiedenen PCs unterschiedlich ist gilt die Zeitskala nur relativ. Aus dieser Darstellung kann entnommen werden, wann der Schüler gearbeitet hat und wie lange ihn welche Fehler aufgehalten haben. So ist z.B. die im Kapitel 5.2.1 erwähnte Unterbrechung der zweiten Sitzung zu erkennen. Außerdem hatte dieser Schüler mehrfach Probleme mit der richtigen Einrückung seines Codes (IndentationError).

Um Rückschlüsse auf Schwächen in der Programmierumgebung ziehen zu können, wurden diese Daten von allen Schülern analysiert. Zunächst wird in Abbildung 16 deutlich, dass die Schüler mit Werten zwischen 71 und 172 sehr unterschiedlich oft den Code ausführten, was aber kaum mit der relativen Häufigkeit von Fehlern korreliert. Prozentual schließen zwischen 60 und 89 Prozent aller Ausführungen ohne Fehler ab. Bei dieser Quote ist zu

bedenken, dass der Code nicht nach jeder Ausführung geändert worden sein muss. Eine besondere Figur in Turtle wurde gerne mehrfach gezeichnet, ebenso wurden fehlerbehaftete Codes mit minimalen, nicht korrekten Anpassungen mehrfach hintereinander ausgeführt. Die hohe Gesamtzahl an Ausführungen lässt sich darauf zurückführen, dass der Prozess in der Programmierumgebung durch einen Button oder eine Tastenkombination sehr schnell starten lässt. Es stellt sich nun die Frage, ob ein höherer Aufwand an dieser Stelle nicht sogar für den Lernerfolg förderlich wäre, da die SuS den Code vermeintlich genauer inspizieren, bevor sie z.B. speichern, kompilieren und erst dann ausführen. Gleichzeitig können durch die Programmierumgebung kleinere Syntaxfehler viel schneller behoben werden und somit die Frustration gering gehalten werden.
Abbildung 17 zeigt nun die Häufigkeit der aufgetretenen Fehler pro Schüler während Abbildung 18 zeigt, wie häufig die verschiedenen Fehlerarten insgesamt relativ auftraten. Zusammengenommen lässt sich erkennen, dass Einrückungs- und Syntaxfehler gemeinsam über die Hälfte der insgesamt aufgetretenen Fehler ausmachen. Außerdem werden bei einzelnen Schülern regelrechte Fehlerpräferenzen deutlich, die einer Lehrkraft zeigen können, wo einzelne SuS Verständnisschwierigkeiten haben. Abbildungen 19 zeigt die

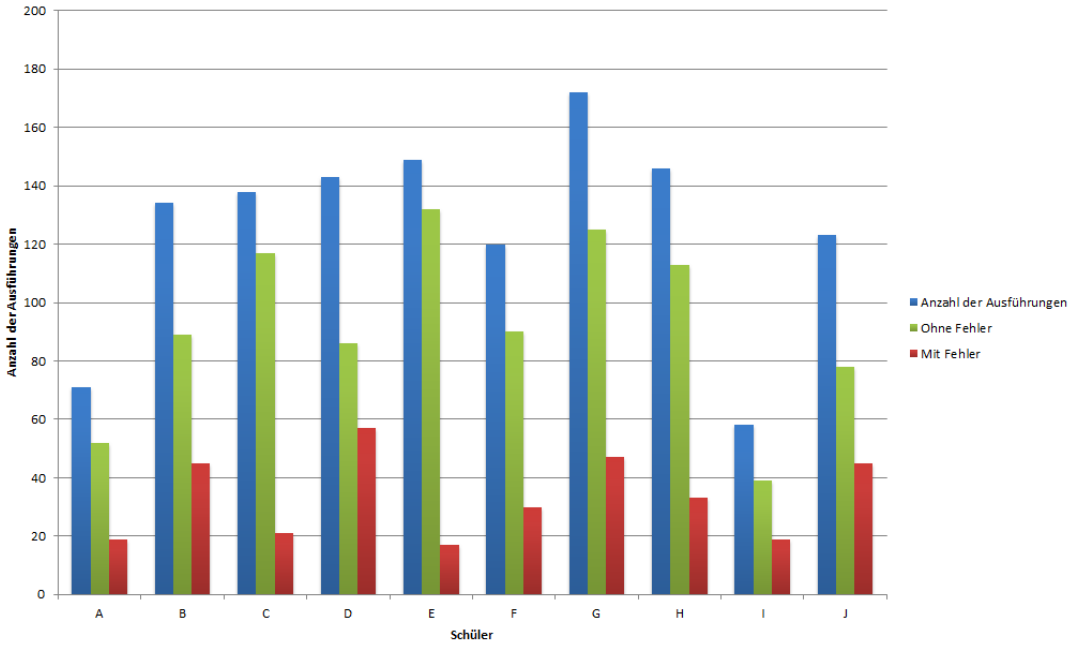


Abbildung 16: Anzahl der Ausführungen gesamt, ohne und mit Fehler.

Dauer zwischen dem Auftreten eines bestimmten Fehlers und dem ersten Ausführen ohne Fehler für jeden Schüler einzeln, während Abbildung 20 den Durchschnittswert über alle Schüler liefert. Aufgrund der relativ geringen Dauer der Benutzerstudie fällt hier das Rauschen sehr groß aus, so ist der vermeintlich schwer zu korrigierende IndexError nur auf Daten eines Schülers zurückzuführen. Die häufiger vorkommenden Syntax- und Einrückungsfehler liefern aber belastbarere Werte.
Die Tatsache, dass weder die Dauer zur Korrektur eines bestimmten Fehlers, noch dessen

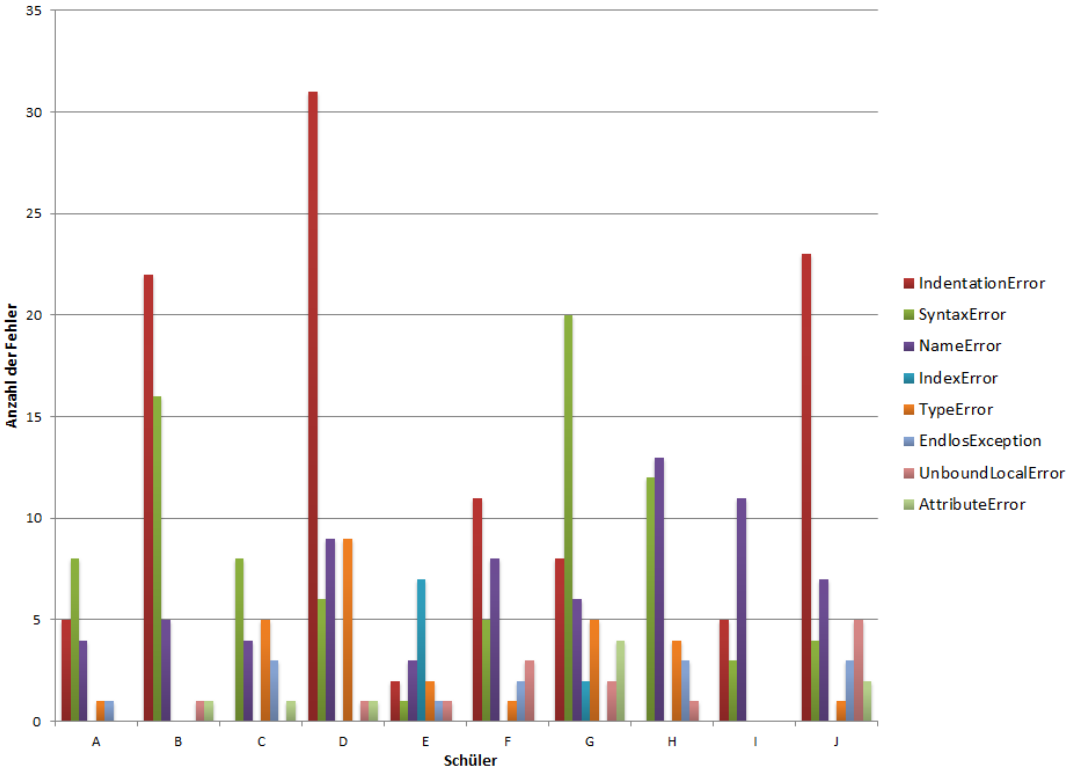


Abbildung 17: Häufigkeit der Fehler pro Schüler und Fehlerart.

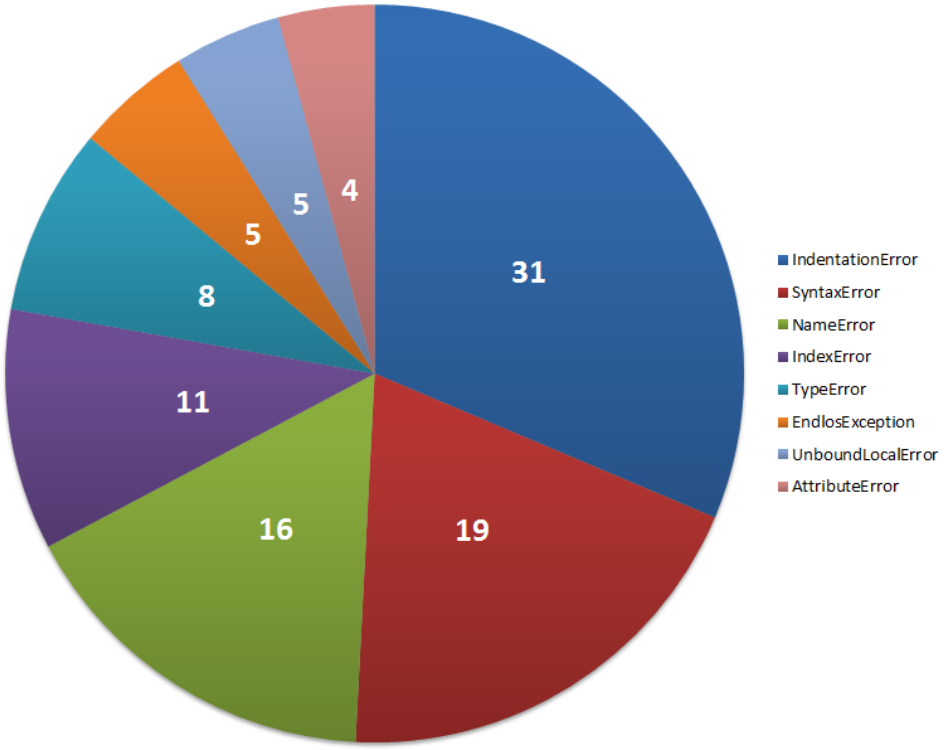


Abbildung 18: Relative Gesamthäufigkeit der Fehlerarten in Prozent.

Häufigkeit die der anderen dominiert, deutet darauf hin, dass die Fehlermeldungen aller Fehlerarten grundsätzlich korrekt ausgegeben werden. Allerdings lässt sich die hohe Dauer

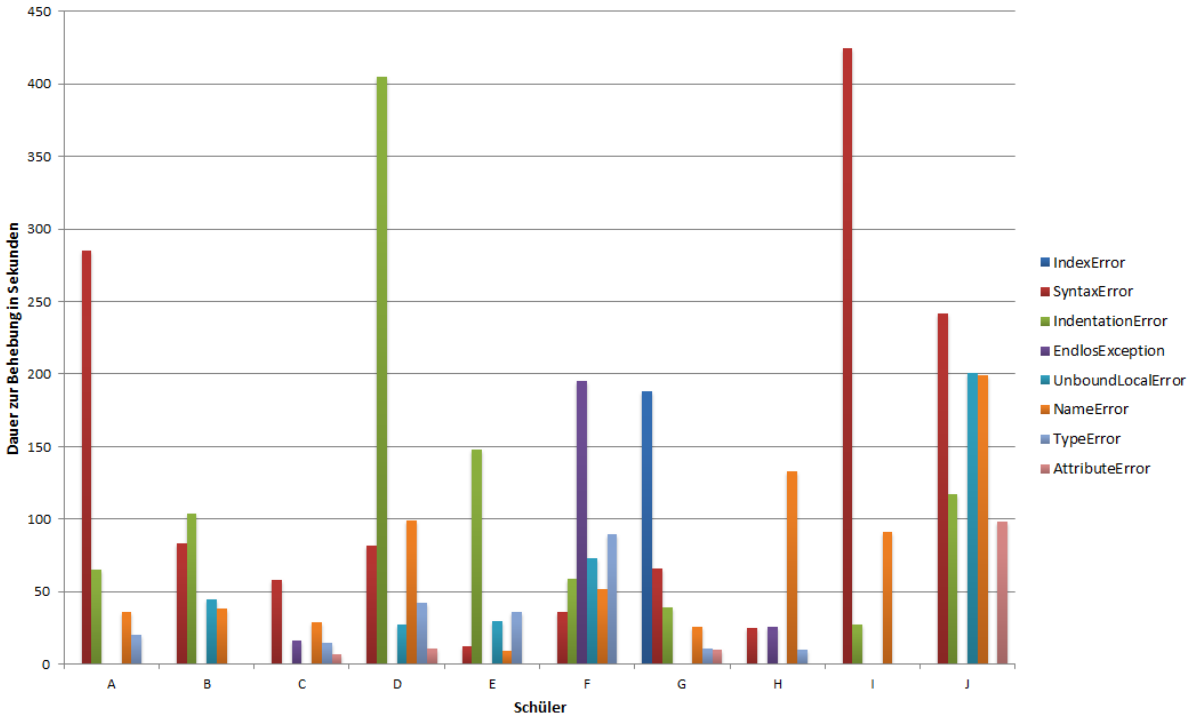


Abbildung 19: Dauer der Fehlerbehebung pro Schüler und Fehlerart.

bei dem vermeintlich am einfachsten zu lösenden Syntaxfehler durchaus auf eine Schwäche der Programmierumgebung zurückführen: Syntaxfehler treten vor der Ausführung des Programms auf, wie in Kapitel 4.2.9 diskutiert ist die Erkennung der betroffenen Zeile dabei nicht immer eindeutig. Problematisch, aber kaum zu vermeiden ist auch die Tatsache, dass z.B. bei einer nicht geschlossenen Klammer die Python-Fehlermeldung auf die Folgezeile verweist, statt auf die tatsächlich fehlerbehaftete. Insbesondere werden Fehler dieser Art in der letzten Zeile des Schülercodes nicht abgefangen, weshalb fälschlicherweise die erste Zeile als fehlerhaft markiert wird.

Rückmeldung des Lehrers

Die vollständige Rückmeldung des Lehrers findet sich im Anhang III. Als Anmerkung wird eine Uhrzeitanzeige, eine Undo-Funktion, ein Hinweis auf die Tastenkombination Ctrl + R zum Ausführen des Programmes und eine Möglichkeit erledigte Aufgaben auszublenden vorgeschlagen. Darüberhinaus wird darauf hingewiesen, dass „die Arbeit der Schüler häufig hinterfragt werden und reflektiert werden“ sollte, da sie sehr selbstständig arbeiten. Diese Ansicht deckt sich mit dem Konzept der Programmierumgebung nachdem sie als Ergänzung zum Unterricht angewendet werden soll.
Insgesamt beurteilte der Lehrer die Programmierumgebung als „ausgereiftes und durchdachtes Hilfsmittel“ mit Verweis auf die Visualisierung und die verständliche Fehlerausgabe.

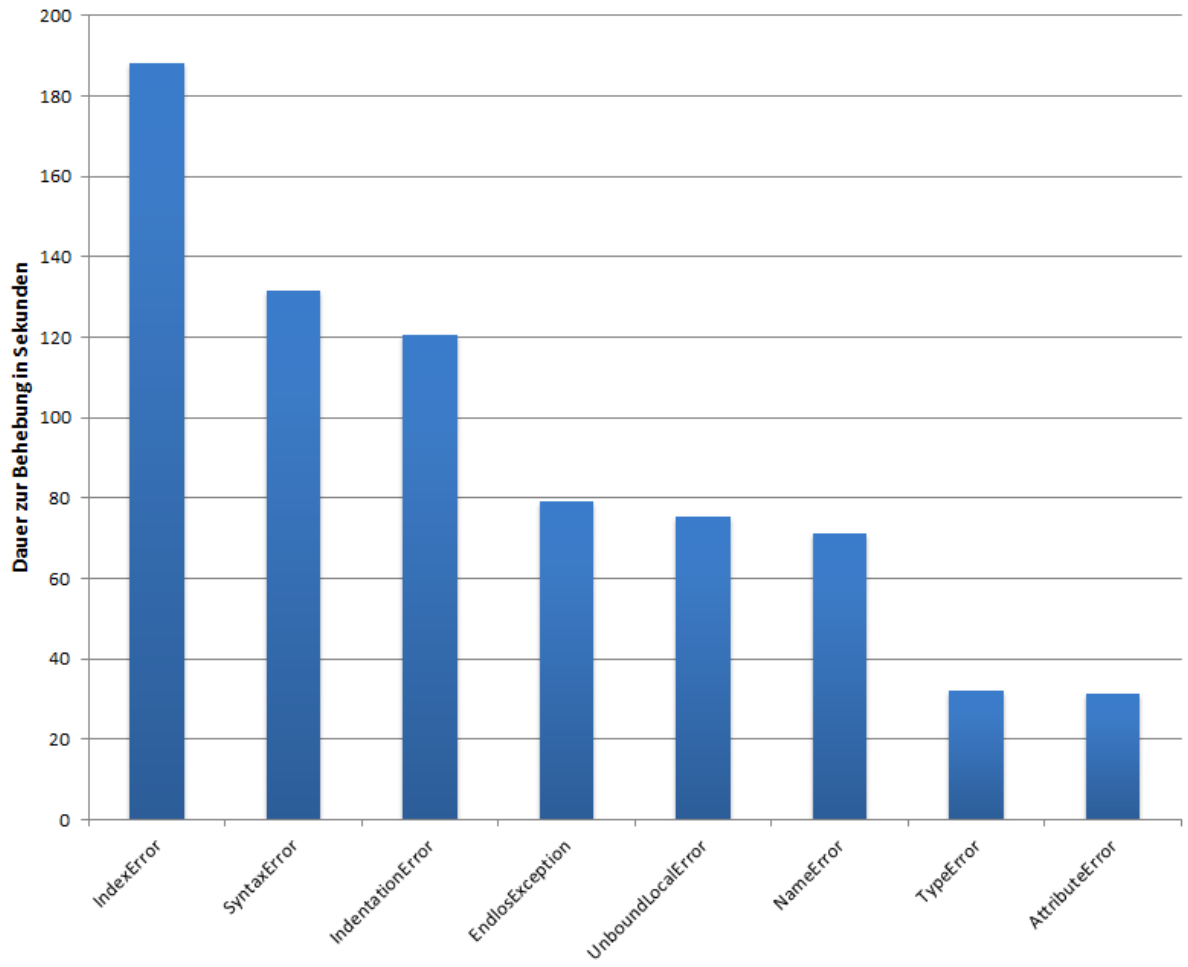


Abbildung 20: Durchschnittliche Dauer der Fehlerbehebung nach Fehlerart. Die hohe Dauer für die Fehlerart IndexError liegt an einem Ausnahmefall eines Schülers.

Qualitative Beobachtungen des Autors

Bei der Durchführung der Benutzerstudie sind zunächst kleinere Bugs aufgetreten. So konnte z.B. die Farbe im Turtle Modul nur durch den RGB Code festgelegt werden, nicht wie angegeben auch über Strings. Programmierte ein Schüler eine Endlosschleife, folgte dann aber nicht dem Rat, das Programm manuell abzubrechen, kam es nach einigen Sekunden zum Absturz des Programms. In späteren Versionen wurde daher die Benachrichtigung durch eine neue Errorklasse `EndlosException` ersetzt, deren Auftreten das Programm wie andere Fehlermeldungen auch beendet. Darüberhinaus ist beobachtet worden, dass Schüler versuchen den Code zu bearbeiten, während er noch im Step Modus ausgeführt wird, was nicht möglich ist. Ein Ausgrauen des Editorfensters während der Ausführung kann hier helfen.

Neben diesen Details funktionierte die Programmierumgebung bei allen Schülern einwandfrei und es kam zu keinen Abstürzen. Die beobachteten Probleme der Schüler ließen sich, bis auf die oben aufgeführte Syntax Fehlermeldung, vollständig auf die eigentliche Aufgabe zurückführen, nicht auf die Programmierumgebung.

Neben der Programmierumgebung spielt die Wahl, die Formulierung und der Schwierigkeitsgrad der gewählten Aufgaben eine große Rolle. Die Aufgaben im Turtle Modul wurden speziell für diese Benutzerstudie entworfen. Ziel war es, möglichst viele Konzepte vorzustellen. Nach der Bearbeitung aller Aufgaben hatten Schüler selbstständig Variablen, If-Entscheidungen, Schleifen, verschachtelte Schleifen, Funktionen mit Argumenten und Iteration über Listen genutzt. Selbst für eine motivierte Schülergruppe ist dieser Inhaltsumfang für 85 Minuten mit 20 Minuten Einführung enorm anspruchsvoll und nach der Einschätzung des Autors ohne eine geeignete Programmierumgebung nicht erreichbar. Die Schüler konnten mit Ausnahme der letzten Aufgabe in der gleichzeitig If-Blöcke, Schleifen und das erste Mal Listen verwendet werden müssen, weitestgehend selbstständig arbeiten. Für die Bewältigung der letzten Aufgabe waren vermehrt Hinweise auf das Handout und Tipps zur Herangehensweise notwendig. Für die Anwendung in einer jüngeren, größeren Klasse muss die Lernkurve dementsprechend flacher gestaltet werden.
Im Gespräch mit Schülern während, zwischen und nach den Stunden wurde von diesen mehrfach erwähnt, dass sie Spaß am Umgang mit der Programmierumgebung hatten. Dies äußerte sich auch in der bei manchen Schülern beobachteten und in der Schule wohl eher seltenen Unwilligkeit, die Arbeit nach Stundenende zu beenden.

Fazit

Die quantitativen Ergebnisse dieser Evaluation sind kritisch zu betrachten. Einerseits war die Menge an Inhalten für drei Schulstunden enorm hoch und die Anleitung und Vorbereitung fiel extrem knapp aus. Andererseits fand die Bearbeitung durch eine kleine Gruppe Oberstufenschüler anstelle der eigentlichen Zielgruppe von Acht- bis Neuntklässlern statt. Zudem wurde die Programmierumgebung als kurzfristiges Tool zur Einführung in eine Programmiersprache verwendet. Dabei ist sie als langfristiges Tool, dass praktische Übungen von theoretischen Inhalten möglich macht konzipiert.
Trotzdem konnte die quantitative Evaluation zeigen, dass keine schwerwiegenden Mängel an der Programmierumgebung existieren, was sich etwa in einer sonst nicht erklärbaren Fehlerart geäußert hätte. Außerdem wurde deutlich, dass Lehrkräfte die im Programm eingebauten Auswertungsmöglichkeiten nutzen können, um Probleme ihrer SuS zu identifizieren. Es konnte außerdem gezeigt werden, dass die Programmierumgebung von Seite der SuS gut angenommen und als valides, motivierendes, wenn auch etwas unansehnliches Werkzeug für den Unterricht gesehen wird, dass Erfolgserlebnisse liefern kann.
Qualitativ wurde die Programmierumgebung nur von einer unabhängigen qualifizierten Person beurteilt. Diese Beurteilung fiel mit der Einschränkung einiger kleinerer Verbesserungsvorschläge positiv aus. Die eigenen Beobachtungen stimmen mit den quantitativen Ergebnissen überein. Die Menge der in kurzer Zeit von Schülern angewendeten und für viele völlig neuen Konzepte ist beeindruckend und spricht für die Schüler ebenso wie für die Programmierumgebung.

6 Ausblick

Eine Verbesserung der Programmierumgebung wäre durch die Implementierung des in Kapitel 4.2 diskutierten Zurück Buttons möglich. Damit würde auch der erwähnte grundsätzliche Umbau von Codeausführung und Animation einhergehen. In jedem Fall sollen die in Kapitel 5.2.2 aufgedeckten Schwächen, insbesondere der Syntaxfehlermeldung, korrigiert werden.
Die Programmierumgebung ist primär für die Verwendung im IMP Profil von Gymnasien erstellt worden. Zu diesem Zweck und mit Blick auf den Lehrplan sind verschiedene weitere Module vorstellbar wie z.B.:

- Mehrere aufeinander aufbauende Module zur Syntaxvermittlung.
- Codierung und Kryptographie.
- Komplexere Graphen und Bäume.
- Bildverarbeitung und -kompression.
- Mehrdimensionale Arrays.
- Datenverarbeitung.
- Fächerübergreifende Module mit Mathematik.
- Fortgeschrittene Module, die nur zur Demonstration fortgeschrittener Konzepte, nicht zur Schülerarbeit gedacht sind.

Darüber hinaus ist auch eine Anwendung der Programmierumgebung in anderen AGs, Klassenstufen oder Bildungseinrichtungen denkbar. Beispielsweise als Einführung in die Programmierung in Klassenstufe sieben mit entsprechenden syntaxlastigen Modulen.
Neben weiteren Modulen sind auch zusätzliche Funktionen geplant:

- Speichern und Upload des individuellen Fortschritts als Rückmeldung oder Bewertungsgrundlage für die Lehrkraft vergleichbar mit Code.org (vgl. Kapitel 3). Notwendig hierfür wäre ein sehr grundlegendes Userverwaltungssystem, in dem die SuS sich über ein zugewiesenes Kürzel anmelden, sowie ein von der Lehrkraft erreichbarer Server mit entsprechendem Backend.
- Vereinfachung der Erstellung von weiteren Modulen und Verfassung einer Anleitung mit dem Ziel, anderen Lehrkräften zu ermöglichen, eigene Module zu schreiben.
- Erweiterung des Codeeditors um bessere Syntaxfehlererkennung, Einrückung mehrerer Zeilen gleichzeitig, Hervorheben selbst definierter Funktionen und Variablen sowie Codevervollständigung.

- Toolbar mit Undo/Redo Button, Speichern, Laden etc.
- Automatische Auswertung nicht nur eines Schülers, sondern einer gesamten Klasse.
- Verbesserung des schlichten Designs.

Der Autor beginnt voraussichtlich mit dem Schuljahr 2018/19 seinen Vorbereitungsdienst am Ganztagsgymnasium Osterburken (GTO). Während dieses Vorbereitungsdienstes und auch im Anschluss soll die Programmierumgebung verbessert und um weitere Module ergänzt werden. Nach der gründlichen Erprobung in der Praxis soll die Umgebung anderen Lehrkräften und Schulen zur Verfügung gestellt werden. Spätestens zu diesem Zeitpunkt soll ein besserer Name als "die Programmierumgebung" feststehen.

Literatur

[1] Mohd Salleh, Syahanim; Shukur, Zarina; Mohamad Judi, Hairulliza (2013), *Analysis of Research in Programming Teaching Tools: An Initial Review*, Procedia - Social and Behavioral Sciences 103, pp. 127-135.

[2] Ministerium für Kultus, Jugend und Sport Baden-Württemberg (2018), *Netzbrief 3 EU-DSGVO*, abgerufen 14.07.2018, https://it.kultus-bw.de/site/pbs-bw-new/get/documents/KULTUS.Dachmandant/KULTUS/Dienststellen/it.kultus-bw/Datenschutz%20an%20Schulen%20nach%20neuer%20EU%20DSGVO/dl-netzbrief/Netzbrief%203%20EU-DSGVO.pdf?attachment=true

[3] Ministerium für Kultus, Jugend und Sport Baden-Württemberg (2018), *Inhaltsübersicht IMP*, abgerufen 14.07.2018, https://www.km-bw.de/site/pbs-bw-new/get/documents/KULTUS.Dachmandant/KULTUS/KM-Homepage/Artikelseiten%20KP-KM/1_PDFS_2016/Inhalts%C3%BCbersicht%20IMP%20(GYM).pdf

[4] Code.org, abgerufen 14.07.2018, https://code.org/

[5] codecademy, abgerufen 14.07.2018, https://www.codecademy.com

[6] Landesinstitut für Schulentwicklung, abgerufen 13.07.2018, http://www.schule-bw.de/faecher-und-schularten/mathematisch-naturwissenschaftliche-faecher/informatik/tools/digitale-schultaschen

[7] Lehrstuhl für Didaktik der Informatik der Universität Duisburg Essen, abgerufen 13.07.2018, https://www.ddi.wiwi.uni-due.de/schule-praxis/links/medien-fuer-den-informatikunterricht/

[8] Gesellschaft für Informatik e.V., Fraunhofer-Verbund IUK-Technologie, Max-Planck-Institut für Informatik: *Bundesweite Informatikwettbewerbe*, abgerufen 13.07.2018, https://www.einstieg-informatik.de/index.php?article_id=866

[9] University of Kent: *Greenfoot 3.1.0*, abgerufen 15.07.2018, https://www.greenfoot.org

[10] Raimond Reichert: *PythonKara*, abgerufen 15.07.2018, http://swisseduc.ch/informatik/karatojava/pythonkara/index.html

[11] R. Reichert, J. Nievergelt, W. Hartmann: *Ein spielerischer Einstieg in die Programmierung mit Java*, Informatik-Spektrum 23 (5), Oktober 2000. Springer Verlag.

[12] Computing Education Research Group - University of Kent: *BlueJ 4.1.2*, abgerufen 13.07.2018, https://www.bluej.org

[13] Lifelong-Kindergarten-Gruppe, Media-Lab, MIT: *Scratch*, abgerufen 14.07.2018, https://scratch.mit.edu/

[14] Philip Guo: *Pythontutor*, abgerufen 12.07.2018, http://pythontutor.com

[15] Stefan Freischlad, Christian Eibl: *Filius 1.7.2*, abgerufen 12.07.2018, http://www.lernsoftware-filius.de

[16] Institut für Wirtschaftsinformatik - Universität Siegen: *Cryptool*, abgerufen abgerufen 16.07.2018, https://www.cryptool.org/

[17] Skulpt, abgerufen 17.05.2018, http://www.skulpt.org/

[18] Developer notes on suspensions for skulpt, abgerufen 17.05.2018, https://github.com/skulpt/skulpt/blob/master/doc/suspensions.txt

[19] Brython, abgerufen 17.05.2018, https://www.brython.info

[20] Python Software Foundation: *GuiProgramming* in Python, abgerufen 17.05.2018, https://wiki.python.org/moin/GuiProgramming

[21] The Python Software Foundation: *Tkinter Documentation*, abgerufen 20.05.2018, https://docs.python.org/3/library/tk.html

[22] Riverbank Computing: *PyQt Documentation*, abgerufen 20.05.2018, http://pyqt.sourceforge.net/Docs/PyQt4

[23] Ministerium für Kultus, Jugend und Sport Baden-Württemberg: *Bildungsplan 2016: Informatik, Mathematik, Physik*, 2018, Stuttgart

[24] Guo, Philip: *Python Is Now the Most Popular Introductory Teaching Language at Top U.s. Universities*, abgerufen 12.05.2018, https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext

[25] Richters, Marcel: *Top 10 der Programmiersprachen: Die aktuellen Rankings im Vergleich*, abgerufen 12.05.2018, https://jaxenter.de/redmonk-pypl-tiobe-rankings-vergleich-69108

[26] Pierre Carbonnelle: *PYPL PopularitY of Programming Language*, 12.07.2018, http://pypl.github.io/PYPL.html

[27] Helmke, A.; Weinert, F. E.:*Bedingungsfaktoren schulischer Leistungen*, in F. E. Weinert (Hrsg.), Enzyklopädie der Psychologie, Band 3 (Psychologie der Schule und des Unterrichts) (S. 71-176). Hogrefe-Verlag, Göttingen, 1997.

[28] Wisniewski, Benedikt: *Psychologie für die Lehrerbildung*. Julius Klinkhardt Verlag, Bad Heilbrunn, 2016.

[29] Wiater, Werner: *Unterrichtsprinzipien*. Auer Verlag, Augsburg, 2018.

[30] Glas, Schneider; Schlagbauer, Johanna: *Pädagogik am Gymnasium*. Cornelsen Verlag GmbH, Berlin, 2017.

[31] Deci, E. L.; Ryan, R. M.: *Intrinsic motivation and self-determination in human behaviour*. Plenum, New York, 1985.

[32] Carr-Chellman, Alison A.: *Instructional-design Theories and Models*. Psychology Press, 1983.

[33] PyInstaller Development Team: *Pyinstaller*. Abgerufen 25.07.2018, http://www.pyinstaller.org

Anhang

Anhang I: Ergebnisse der offenen Fragen des Evaluationsbogens

Sind dir bei der Programmierumgebung Fehler aufgefallen, wenn ja welche genau?

"Die Aufgabe 5 muss abgeändert werden.(Rechteck)"

"Anzeigefenster zu klein"

"Ja, wenn die # nicht am Zeilenanfang stehen ist das ein Fehler"

Was hat dir an der Programmierumgebung gefallen?

"Die Visualisierung und der einfache Umgang"

"Die Umgebung ist übersichtlich und anfängerfreundlich gestaltet"

"sehr übersichtlich, Aufgaben waren anspruchsvoll"

"Dauer-pro-Schritt-Funktion "

"Übersichtlich und Kinderfreundlich gestaltet "

"leicht verständlich, übersichtlich"

"Die Erklärung bzw. Ausführung der Aufgabe im linken Fenster "

"Programmieren zu verstehen und selber was zu erstellen "

"Das sehr anschaulich ist und relativ deutlich "

"anschaulich, einfach zu verstehen, übersichtlich "

Was hat dir an der Programmierumgebung nicht gefallen?

"Sie sieht nicht sehr hübsch aus "

"Die Zeilennummerierung wird leicht überdeckt "

"Anzeigefenster lässt sich nicht vergrößern "

"einfaches Design (zu einfach) "

"Manche Fehlercodes teils unverständlich "

Was würdest du an der Programmierumgebung ändern?

"Bis auf Design nichts "

"Das Aussehen"

"zurück Befehl, mehrere Zeilen gleichzeitig einrücken"

"Den Programm Schließen Knopf nicht in der nähe des Ausführen Knopfes"

"außer dem Aussehen eigentlich eine grundsolide Umgebung "

Anhang II: Bewertung der Inhalte der IMP

Bezeichnung	Klasse	Bemerkung	Mögliche Umsetzung	Bewertung (1-5)
Fehlererkennung / Korrektur	8		Validierung von ISBN, Kreditkarte	2
Listen	8	Aufbau und grundlegende Operationen	Einfache Operationen, Liste erstellen, anhängen, einzelne Elemente abrufen	4
Bäume	8		Bäume erstellen und zeichnen	5
Graphen	8		Graphen erstellen und zeichnen	5
Programmieren in visueller Umgebung	8	Definition von visueller Umgebung unklar: textuelles oder graphisches Programmieren?		x
Unterprogramme	8		Programm mit mehreren unterfunktionen ausführen lassen (evtl. mandala zeichnen)	4
Programmierprojekt	8	Ist in andere Module "eingebaut", alternativ kann auch ein freieres Programm geschrieben werden		x
Rechnernetz	8	theoretische Behandlung besser geeignet		x
Adressierung im Netzwerk	8	theoretische Behandlung besser geeignet		x
Simulation von Rechnernetzen	8	Das Programm Filius ist hier besser geeignet		x
WWW	8	theoretische Behandlung besser geeignet		x
Verschlüsselung	8	theoretische Behandlung besser geeignet		x
Prinzipien der Kryptographie	8	theoretische Behandlung besser geeignet		x
Praktische Anwendung von Verschlüsselung	8		Viginiere / OneTimePad / Cäsar übersetzer selbst schreiben	2
Personenbezogene Daten	8	theoretische Behandlung besser geeignet		x
Digitalisierung analoger signale	9	möglich aber wohl eher langweilig?	"signal" wird vorgegeben und muss verarbeitet werden, Ergebnis wird automatisch verglichen?	2
Datenkompression	9	möglich aber wohl eher langweilig?	schnell recht komplex aber: Schwarz weiß Bild komprimieren	2
textuelle Programmierung	9	In anderen Modulen enthalten		5
Datentypen	9		Vorstellung und Umwandlung verschiedener Datentypen	3
Algorithmen auf Arrays	9		Sortieren	5
Algorithmus von Dijkstra	9		zunächst BFS und DFS dann Dijkstra	5
Asymmetrische Verschlüsselung	9	Grundlagen aus Mathe		1
Signierung Zertifizierung	9	theoretische Behandlung besser geeignet		1
Mehrdimensionale Arrays	10		Listen von Listen	3
Algorithmen zur Bearbeitung von Bitmaps	10		"live" Bildmanipulation (z.b. Graustufenmatrix)	4
Interaktive Anwendungen mit grafischer Benutzerschnittstelle	10		Anwendung von Schülern über vereinfachtes PyQt innerhalb der Programmierumgebung erstellen lassen. Visualisierungsbereich ersetzen durch das SchülerGUI	2
Gatter, Schaltnetze, Wahrheitstafeln	10	theoretische Behandlung besser geeignet		x
Rechneraufbau, Addierer, Flipflop	10	theoretische Behandlung besser geeignet		x
Adressierung und Routing in Netzen	10	theoretische Behandlung besser geeignet		x
Datenverlust,	10	theoretische Behandlung besser geeignet		x
Datensicherung	10	theoretische Behandlung besser geeignet		x

Anhang III: Bewertung der Programmierumgebung von Christian Schinnagel

GANZTAGSGYMNASIUM
OSTERBURKEN

74706 Osterburken,
Hemsbacher Str. 24

Telefon 06291 64080
Fax 06291 640813

Ganztagsgymnasium - 74706 Osterburken

Anmerkungen zur Benutzerstudie der Lernumgebung von Yannik Ries

Herr Ries führte im Zeitraum vom 24. September bis zum 8. Oktober eine Benutzerstudie seiner selbst entwickelten Lernumgebung am Ganztagsgymnasium Osterburken durch. Dies beinhaltete drei Schulstunden im Informatik-Kurs der Jahrgangsstufe 1 bzw. 2 sowie ein vorbereitendes Treffen. In der letzten Stunde wurde außerdem eine Evaluation durchgeführt.

Der Informatikkurs besteht aus insgesamt neun Schülern, wovon vier schon in der Jahrgangsstufe 2 sind. Der Kurs hat stets in einem der Computerräume des Ganztagsgymnasiums Unterricht. Hier steht jedem Schüler ein PC zur Verfügung, außerdem gibt es eine Tafel und einen Beamer. Der Kurs ist sehr inhomogen: einige der Schüler haben schon Programmiererfahrung, andere noch gar keine. Der zweistündige Kurs findet Montags und Mittwoch vormittags statt. Die Schüler wurden innerhalb der Stunden von Herrn Ries betreut, der Fachlehrer unterstützte während der Übungsphase.

Installation/ Vorbereitung: Die Lernumgebung benötigt keine Installation und lässt sich leicht innerhalb der verwendeten Infrastruktur (Server mit der Linuxmuster.net Software mit Windows-Clients) an die Schüler verteilen.

Verlauf der Stunden: In der ersten Stunde wurde eine kurze Einführung in die Sprachkonzepte von Python gegeben, anschließend arbeiteten die Schüler weitestgehend selbstständig an den Aufgaben aus der Lernumgebung. Gleich zu Beginn zeigte sich, dass die Schüler keine Probleme hatten die Lernumgebung zu starten und zu benutzen. Einige Schüler verbrauchten etwas Zeit zum ständigen Klicken auf die "Ausführen" Taste, hier wäre ein Tooltip o.ä. an dem Button wünschenswert.

Im Allgemeinen arbeiteten die Schüler bemerkenswert selbstständig an den gestellten Aufgaben, unabhängig vom Grad der Vorerfahrung. Durch die

allgemeinverständlichen Fehlermeldungen musste die Hilfe der Lehrkraft nur selten in Anspruch genommen werden.

Durch das hohe Maß an selbstständiger Arbeit sollte die Arbeit der Schüler häufig hinterfragt werden und reflektiert werden. Durch die Autosave-Funktion ist der Lehrkraft hier eine Möglichkeit an die Hand gegeben, bei der zur Reflektion die Probleme und Erfahrungen aus den vorherigen Stunden herangezogen werden können. Außerdem kann die Lehrkraft so auch mit den Schüler üben nicht nur richtigen, sondern auch gut verständlichen Code zu schreiben, indem Beispiele für die Klasse herausgesucht und verbessert werden.

Anmerkungen: Die Lernumgebung startet standardmäßig im Vollbildmodus und verdeckt dabei die Taskleiste von Windows. Was hilfreich ist die Ablenkung zu minimieren, hat den Nachteil, dass die Uhr verdeckt wird. Den Schülern hilft es aber die verbleibende Zeit bis zum Stundenende einschätzen. Eine einfache Möglichkeit diese Verhalten zu beeinflussen wäre hier hilfreich.

Zum Teil vermissten die Schüler eine "Rückgängig"-Funktion. Die Tastenkombination Strg Z funktioniert zwar, allerdings kennen gerade die Schüler mit wenig Erfahrung diese nicht. Hier könnte man die Funktion vielleicht noch etwas mehr herausstellen.

Da die Aufgabentexte am linken Rand teilweise sehr lang sind, wäre es zudem schön, wenn man bereits bearbeitete Aufgaben "einklappen" könnte, so dass mehr Text in das Fenster passt.

Fazit: Insgesamt präsentierte sich die Lernumgebung in den drei Stunden als ausgereiftes und durchdachtes Hilfsmittel beim Erlernen der Programmiersprache Python. Hervorzuheben sind insbesondere die Möglichkeit dem Programm beim Abarbeiten des Codes "zuzusehen" als auch die leicht verständlichen Fehlerausgaben. Die motivierenden Aufgaben boten genügend Möglichkeiten, um sich mit den Konzepten der Sprache auseinander zu setzen. Später kann in der gleichen Umgebung an anderem Stoff (wie etwa Graphentheorie und Sortieralgorithmen) weitergearbeitet werden.

Anhang IV: Handout zur Python-Syntax

Mini Python Crashkurs

Ausgabe und Kommentare

Normalerweise wird Python direkt über die Kommandozeile ausgeführt, Ausgaben des Codes werden dann über den print() Befehl in diese Kommandozeile "gedruckt". In der Programmierumgebung erscheint die Ausgabe im Ausgabefenster.

`print(5)`	Ausgabe: 5
`print(5+2)`	Ausgabe: 7
`print("hallo")`	Ausgabe: hallo
`# print("hallo")`	Keine Ausgabe. Das # signalisiert ein Kommentar, das keinen Einfluss auf das Programm hat.

Variablen

`a = 5`	Hier ist in der Variable a die ganze Zahl (engl: Integer) 5 hinterlegt.
`variable2 = 7-3`	Hier ist in Variable2 die 4 hinterlegt.
`fertig = True`	Hier ist der booleschen Variable der Wert True zugewiesen. Boolesche Variablen können nur entweder True oder False sein.
`ergebnis = a + variable2`	Hier ist in ergebnis die 9 hinterlegt.
`text = "Hallo Welt"` `print(text)`	Hier ist text der Text (engl. String) Hallo Welt zugewiesen. Die Anführungszeichen markieren die Worte als String, sonst könnten sie als Variablennamen missverstanden werden. Ausgabe: Hallo Welt

Entscheidungen

Wenn ein Codeblock nur unter bestimmten Bedingungen ausgeführt werden soll, verwendet man if-Abfragen:

`if a > b:`
`  print("a ist größer b")`
`elif a < b:`
`  print("a ist kleiner b")`
`elif a == b and b != c:`
`  print("a ist gleich b und b ist ungleich c")`
`else:`
`  print("keiner der obigen Fälle ist eingetreten")`
	Der print() Befehl wird jeweils nur ausgeführt, wenn die abgefragte Bedingung erfüllt ist. Sobald eine Bedingung erfüllt ist wird NUR der darunter stehende Code ausgeführt. Nachfolgende elif und else Abfragen werden übersprungen. Man beachte den Doppelpunkt und die Einrückung der Zeilen nach einem if, elif, oder else.

`==, !=, <=, >=, <, >`	Verschiedene mögliche Vergleiche: gleich, ungleich, kleiner oder gleich, größer oder gleich, kleiner, größer.

Schleifen

Wenn ein Codeblock mehrfach ausgeführt werden soll, werden for oder while Schleifen verwendet:

`counter = 0`
`while counter <= 5:`
`  print(counter * counter)`
`  counter = counter + 1`
	While-Schleifen eignen sich besonders, wenn man nicht genau weiß, wie oft etwas ausgeführt werden soll.
	Nicht vergessen: Doppelpunkt und Einrücken nach while.
	Die Ausgabe wäre hier: 0, 1, 4, 9, 16, 25

`for counter in range(5):` `  print(counter)`	Die Ausgabe wäre hier: 0, 1, 2, 3, 4 Nicht vergessen: Doppelpunkt und Einrücken nach for.
`for buchstabe in ["a","b","c"]:` `  print(buchstabe)`	For-Schleifen eignen sich besonders gut für das Arbeiten mit Listen. Die Ausgabe wäre hier: a, b, c

Funktionen

Wenn ein bestimmter Arbeitsschritt an verschiedenen Stellen in deinem Code mehrfach auftritt solltest du eine Funktion schreiben, die diesen Arbeitsschritt ausführt, das macht deinen Code übersichtlicher und später auch mächtiger.

```
def blabla():
    print("das ist doch alles
           nur Gelaber")
```

Hier wird die Funktion blabla definiert. Beim Ausführen passiert zunächst nichts, denn die Funktion ist definiert, wird aber nicht aufgerufen.

```
def blabla():
    print("das ist doch alles
           nur Gelaber")

blabla()
```

Hier wird die Funktion auch aufgerufen.
Die Ausgabe wäre: das ist doch alles nur Gelaber

```
def addieren(a,b):
    ergebnis = a + b
    return ergebnis

print(addieren(5,2))
```

Du kannst Funktionen auch Argumente übergeben. Hier werden von der Funktion addieren zwei Argumente erwartet. Das Erste wird dann mit a bezeichnet, das Zweite mit B. Außerdem wird mit dem return Befehl die Variable ergebnis zurückgegeben, wann immer die Funktion aufgerufen wird.
Die Ausgabe wäre hier: 7

Arrays / Listen

In Python werden Arrays einfach als Listen bezeichnet. In einem Array sind Elemente nacheinander angeordnet. Jedes Element hat einen Index entsprechend seiner Position. Elemente können dabei Zahlen, Text, andere Arrays oder andere Objekte sein.

```
zahlen = [1, 4, 7, 10]
print(zahlen[0])
print(zahlen[2])
print(zahlen[-1])
```

Hier wird das Array zahlen erstellt. Danach werden einzelne Elemente durch ihren Index aufgerufen und gedruckt. Beachte, dass Arrays immer mit dem Index 0 beginnen!
Falls der Index negativ ist, wird vom Ende des Arrays her gezählt, d.h. das letzte Element hat den Index -1. Die Ausgabe wäre hier: 1, 7, 10

```
zahlen = [1, 4, 7, 10]
zahlen[2] = 5
print(zahlen[2])
```

Elemente eines Arrays können auch überschrieben werden.
Die Ausgabe wäre hier: 5
Für mehr Operationen auf Arrays bearbeite das Modul Listen in der Kategorie Listen.

Datentypen

Es kann manchmal notwendig werden, Datentypen ineinander überzuführen.

```
print(1 + "2")
```

Hier tritt ein Error auf, weil Python nicht Zahlen (integer) und Text (strings) addieren kann.

```
print("1" + "2")
```

Addition von Strings ist bei Python einfach nur das Aneinanderhängen.
Ausgabe: 12

```
print(1 + int("2"))
```

Der int() Befehl konvertiert einen string in einen integer.
Ausgabe: 3

```
string = str(125)
print(string[0])
print(string[1])
print(string[2])
```

Der str() Befehl konvertiert integer in strings. Einzelne Zeichen eines strings lassen sich aufrufen wie Elemente eines Arrays
Ausgabe: 1, 2 ,5

Anhang V: Evaluationsbogen

(optional) Dein Benutzername im Schulsystem: _____

Vorkenntnisse	trifft gar nicht zu	trifft eher nicht zu	trifft eher zu	trifft voll zu
Ich habe bereits Programmiererfahrung.				
Ich habe bereits Erfahrung in Python.				
Ich kenne grundlegende Konzepte der Programmierung wie Variablen, Schleifen und Funktionen bereits.				

Programmierumgebung	trifft gar nicht zu	trifft eher nicht zu	trifft eher zu	trifft voll zu
Ich habe direkt mit dem Programmieren angefangen, statt die Einführung zu hören				
Ich hatte Erfolg beim Programmieren.				
Ich wusste immer, was von mir verlangt wird.				
Die Visualisierung der Liste hat mir geholfen Fehler zu finden.				
Die Visualisierung der Liste hat mir geholfen zu verstehen wie Listen funktionieren.				
Die Visualisierung von Turtle hat mir geholfen Fehler zu finden.				
Die Visualisierung von Turtle hat mir geholfen zu verstehen wie Variablen, Schleifen und Funktionen funktionieren.				
Die Fehlermeldungen haben mir geholfen, meinen Fehler selbst zu finden.				
Ich habe mein Programm oft Schritt für Schritt ausgeführt				

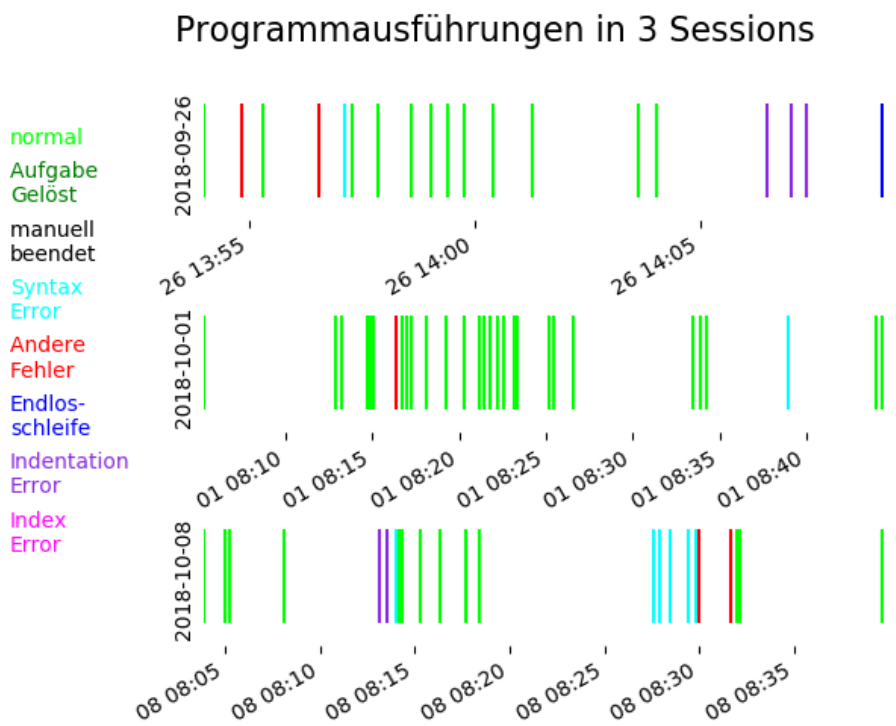
Sind dir bei der Programmierumgebung Fehler aufgefallen? Wenn ja welche genau?

Was hat dir an der Programmierumgebung gefallen?

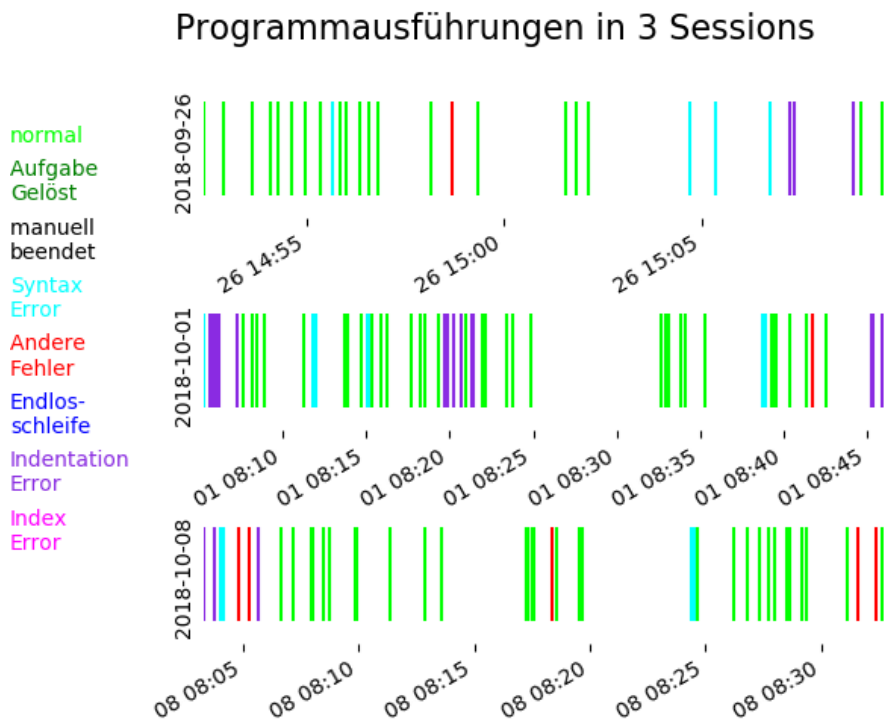
Was hat dir an der Programmierumgebung nicht gefallen?

Was würdest du an der Programmierumgebung ändern?

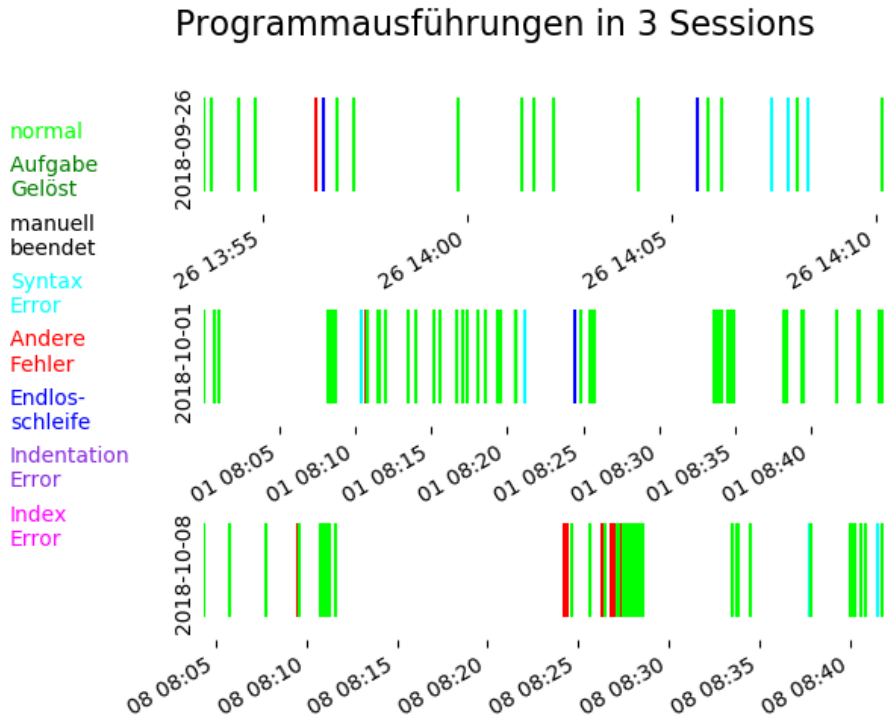
Anhang VI: Auswertung des Schülercodes



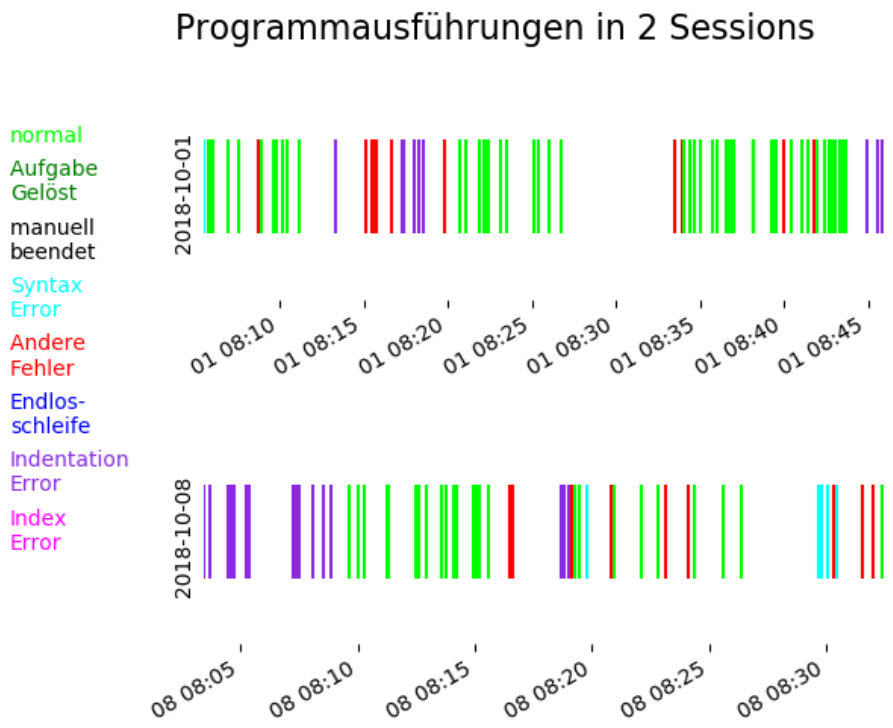
Auswertung des Schülers A



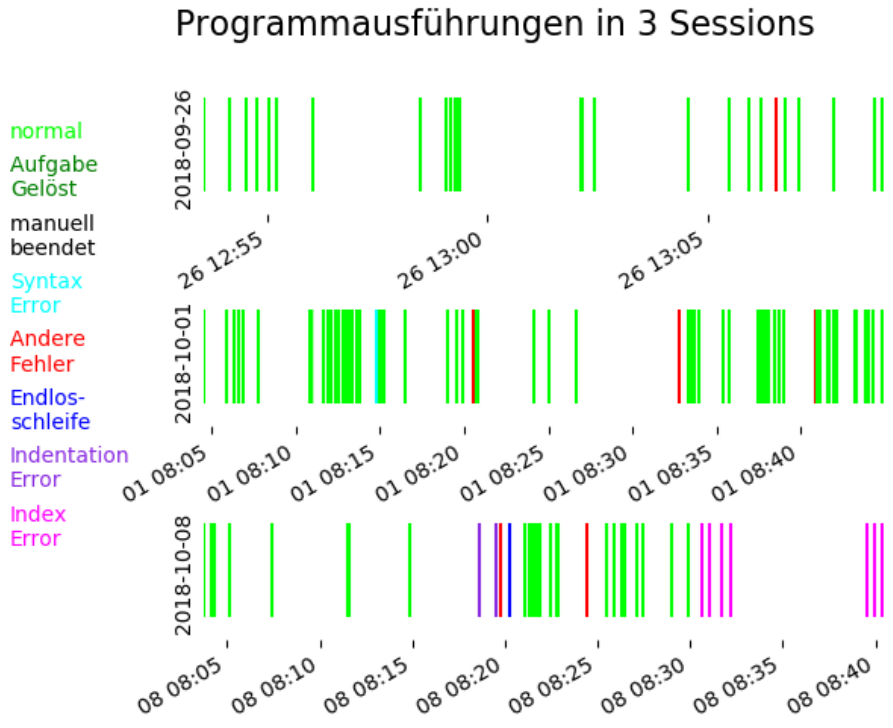
Auswertung des Schülers B



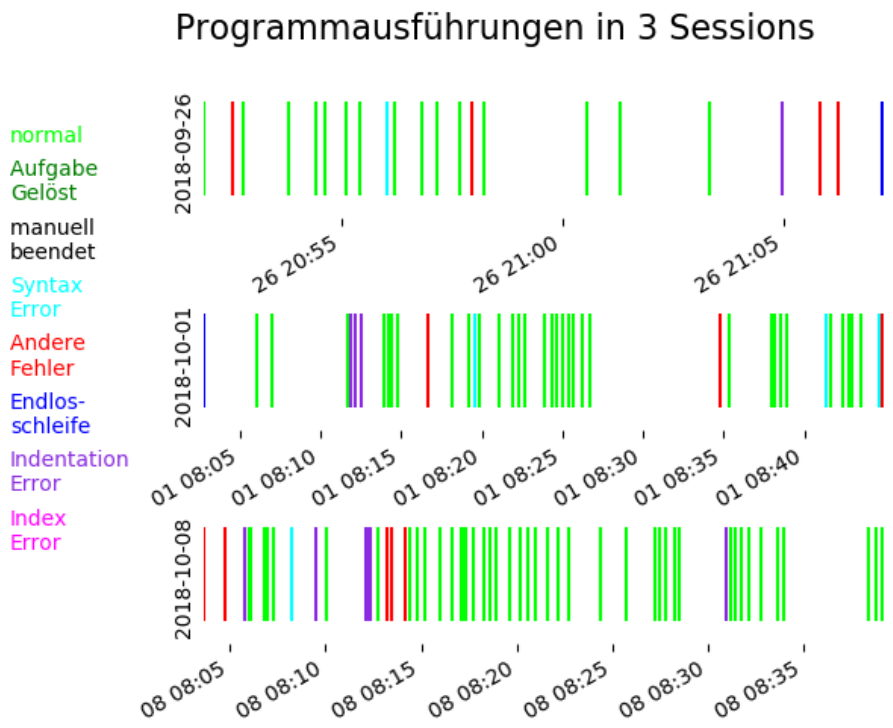
Auswertung des Schülers C



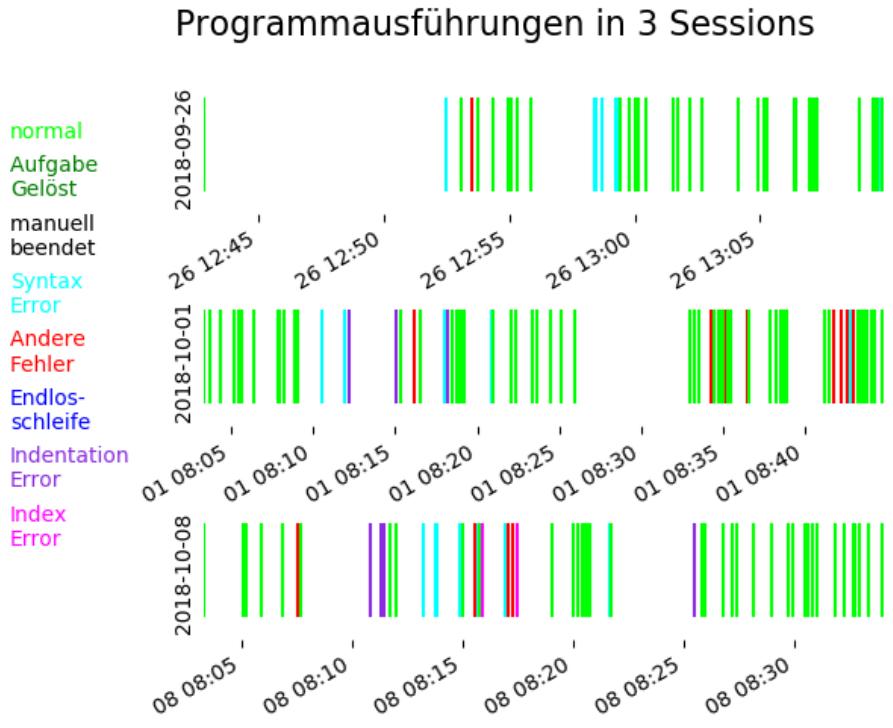
Auswertung des Schülers D



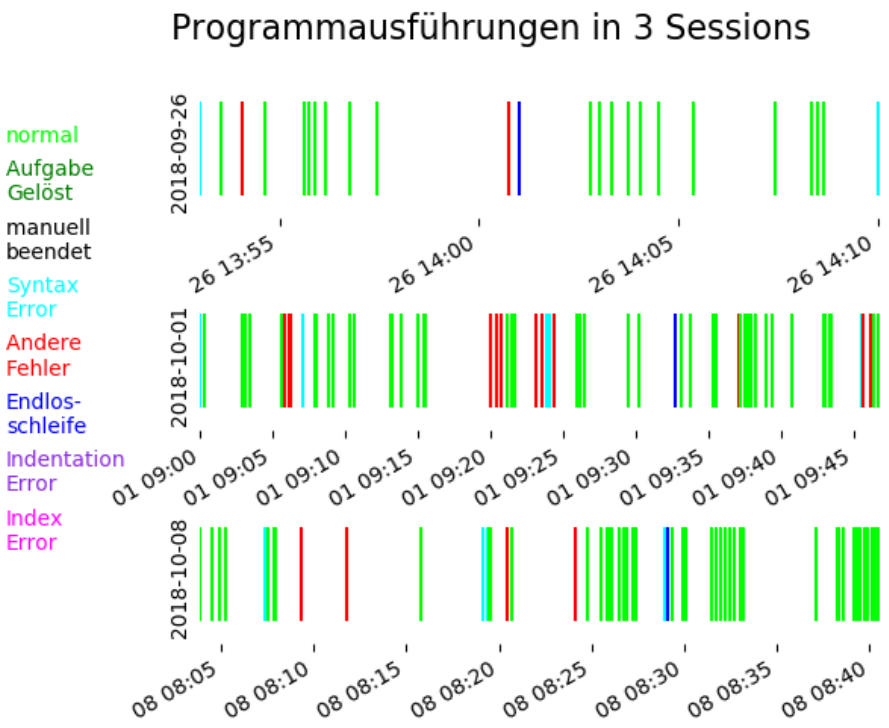
Auswertung des Schülers E



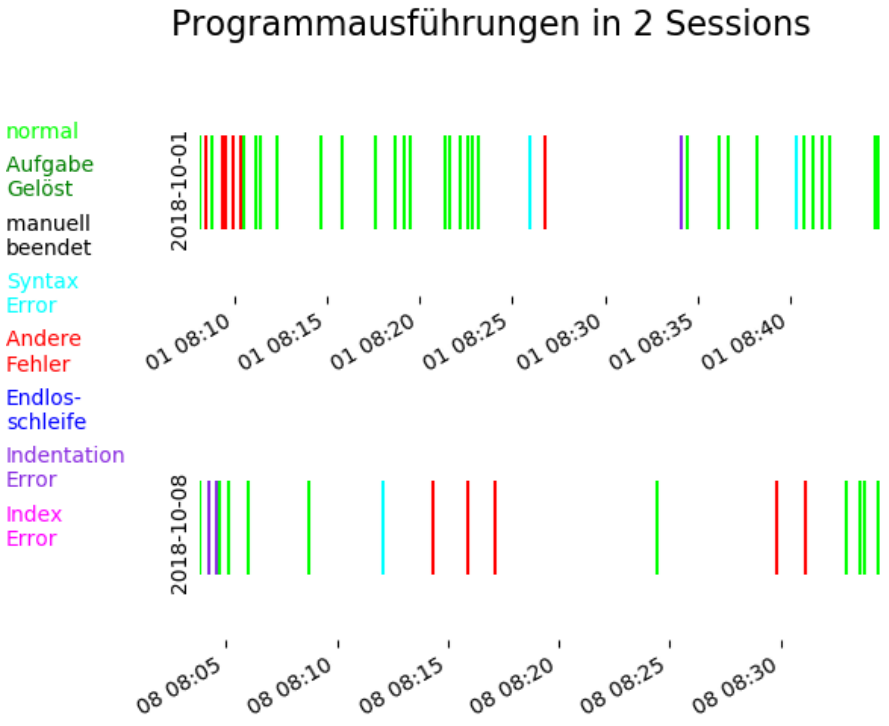
Auswertung des Schülers F



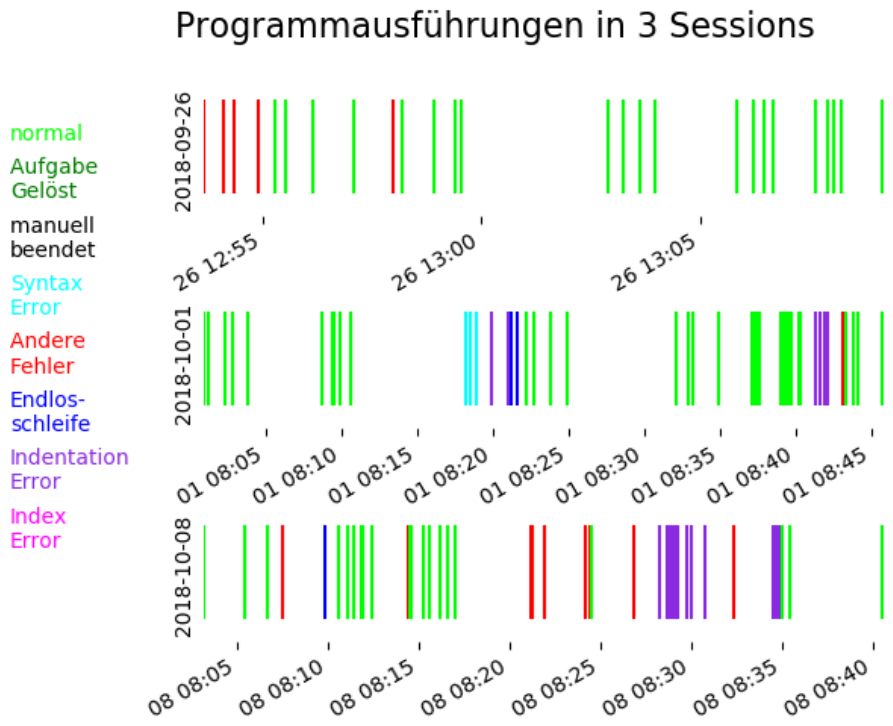
Auswertung des Schülers G



Auswertung des Schülers H



Auswertung des Schülers I



Auswertung des Schülers J