FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Erlernen von Programmieren an Oberstufen der Gymnasien Österreichs durch Computergrafik-unterstützte Ausgabe

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Software- und Informationengineering

eingereicht von

Mag. Christian Johannes Tomaschitz, BSC
Matrikelnummer 00051491

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Assoz. Prof. DI DI Dr. Michael Wimmer

Wien, 22.01.2020

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Erklärung zur Verfassung der Arbeit

Mag. Christian Johannes Tomaschitz, BSC
Terrassenwohnpark 106, 7082 Donnerskrichen

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

---------- ----------

(Ort, Datum) (Unterschrift Verfasser/in)

Danksagung

Ich bedanke mich sehr bei all jenen, welche mich bisher auf meinem Weg begleitet haben und welche für mich in den letzten Jahren dagewesen sind. Den Abschluss des TU Studiums mit der Masterarbeit verdanke ich meiner Verlobten, die mich zum Abschließen meines Studiums bewegt hat.

Kurzfassung

Das Lernen von Programmieren ist für Anfänger eine Herausforderung. Verschiedene Fähigkeiten und Fertigkeiten müssen miteinander kombiniert werden, um die Kompetenz Programmieren erlangen zu können. Die geforderten Fähigkeiten für erfolgreiches Verständnis des Programmierens sind mathematischer, logischer und EDV-technischer Natur. Zu den EDV Fertigkeiten zählen der Umgang mit dem Computer und die Beherrschung einer Programmiersprache.
Der Prozess des Lernens einer Programmiersprache und die Entwicklung der Denkweise ist ein Prozess, der neben dem regelmäßigen Auseinandersetzen mit der Thematik ausreichend Zeit zum Verarbeiten erfordert. In den Lehrbüchern wird Programmieren mit einer Programmiersprache vermittelt und wirkt für Einsteiger meist sehr komplex und trocken. Ein Grund für diese trockene Wirkung ist das textbasierte Arbeiten, welches in den meisten Lehrbüchern gezeigt wird.
In den letzten Jahren stellten sich zum klassischen Frontalunterricht andere motivierende Faktoren, wie aktive Gestaltung des Unterrichts oder Förderung unterschiedlicher Sinne beim Lernen als förderlich heraus. Eine zentrale Rolle nimmt aktives Lernen und eine Kombination aus auditivem und visuellem Lernen ein. Speziell die visuelle Aufbereitung in der Vermittlung von Wissen ist förderlich bezüglich der Lerneffizienz.
An Österreichs Oberstufen der Gymnasien ist für die Anzahl der zu vermittelnden Lehrinhalte die Zeit knapp bemessen und Lerneffektivität ist bei SchülerInnen gefordert, um in kurzer Zeit eine stark ansteigende Lernkurve zu generieren.
Aus diesem Grund ist eine Methode gefordert, die dieses Vorhaben in der Vermittlung der Kenntnisse im Programmieren effektiv unterstützt. Aus persönlicher Erfahrung in der Programmierung ist das Begreifen von abstrakten Lehrinhalten durch Visualisierung einfacher und effektiver. Dies führt zur hier vorgestellten Methodik zum Lernen des Programmierens durch visuell basierte Ausgabe zur Beschleunigung in der Vermittlung von Wissen.

In dieser Arbeit werden zu Beginn aktuelle Forschungen und Grundlagen für die vorgestellte grafisch basierte Lernmethodik des Programmierens erarbeitet. Im Anschluss wird ein Vergleich zwischen der textbasierten „klassischen“ und grafisch basierten „grafischen“ Methodik gezogen und die Ergebnisse beschrieben. Am Ende werden die Ergebnisse konsolidiert und die Effektivität der „grafischen“ Methode beschrieben.

Keywords: *Lernmethode, Lerneffizienz, Programmieren, textuell, visuell*

Abstract

Learning programming is a challenging task as several skills, have to be learned. Some of those are mathematical knowledge, logical knowledge and knowledge about handling the computer. By having these skills, someone could start learning programming in a more or less effective way. However, the skill „programming“ describes the way of thinking someone should achieve, not only writing correct code with the help of a programming language.

But learning to think like a programmer needs time, as described above. Many different skills for solving software problems are needed. Therefore the way of learning is essential. Many books teaching how to program only show the right way of using a programming language and do not focus on the essential part: the way of thinking.

Another problem shown in past researches is based on the way of teaching. Motivated by the right factors, student could achieve better results when learning new content. Using the method of active learning with combined audio and video sources for teaching caused much more effective learning results as humans are trained to remember faster and longer what they have seen or heard.

In Austria, students are forced to learn a variety of clearly seprated topics within three years to gain knowledge about informatics before leaving high school. Breaking it down, the time for learning programming basics within a year are about 8 weeks, having 2 units à 50 minutes. So time is short and the teacher is forced to use an effective way to teach programming and show students how to think like a programmer.

Having all these facts in mind, a method shown in this document is developed that is effective enough to achieve the goal of an effective learning curve. As personally experienced, learning programming is much easier having the right way of representing what is going on. The „classic“ way of showing what is going on is by having textual output, while the method developed here uses graphical output to represent what has been programmed. The aim of this research is to analyse the effectivness of graphical output over textual output.

The structure of the document is as follows: analysis of actual research, basics used in the learning method developed and comparison between programming with textual output and programming with graphical output. In the end, the results are shown and a perspective is shown.

Keywords: *learning method, learning efficiency, programming, text based, visual based*

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Programmieren zu lernen ist für viele SchülerInnen eine Herausforderung. Es müssen unterschiedliche Aspekte wie Variablen, Eingabe- und Ausgabedaten beziehungsweise Algorithmen verstanden werden, bevor ein Programm von einem Schüler, einer Schülerin selbständig programmiert werden kann [1].

Eine geeignete Vermittlung von Wissen basierend auf Theorie und Praxis ist die zentrale Komponente eines Unterrichts und Herausforderung zugleich [2]. Die Pädagogik unterstützt den Lehrenden hier mit verschiedenen Theorien, um einem Schüler beziehungsweise einer Schülerin gezielt Wissen zu vermitteln und Kompetenz in einem Bereich zu erlangen. Verschiedene Ansätze beschäftigten sich dabei mit aktivem Lernen und wie dadurch Wissen beim Schüler, bei der Schülerin entsteht [3,4]. Durch Schaffung eines geeigneten Lernumfeldes und Veränderung des klassischen Schülerbildes im Unterricht vom Zuhörer hin zum aktiven Mitglied mit der Möglichkeit Fehler zu machen, ohne dafür Konsequenzen zu erwarten, soll Wissen dem Schüler, der Schülerin effektiver vermittelt werden [4]. Ein Bereich des aktiven Lernens basiert auf der Erarbeitung von Wissen mittels aktiven Schreibeinheiten, kombiniert mit Simulationen und Spielen, teils auch kooperativ und mit visuellem Lernen [3]. Die Effektivität von aktiven Lernmethoden, manuell, spielerisch und visuell sind in unterschiedlichen Bereichen des Lernens immer wieder aufgezeigt worden [6].

Einem Schüler, einer Schülerin Programmieren beizubringen beinhaltet viele Elemente aktiven Lernens: Schreiben von Code, spielerisches Simulieren von Abläufen und Prozeduren (Algorithmen) sowie Teile der visuellen Repräsentation und Aufnahme durch den Schüler, der Schülerin bei der Ansicht der Ausführung des Programmablaufs. Der letzte Aspekt, die Ansicht des Programmablaufs, ist ein wesentlicher Punkt bei der Erlangung von Wissen.

Bei den klassischen Methoden zur Vermittlung von Programmierkenntnissen an Schulen wird mittels textbasiertem Eingabe- und Ausgabewerk (Texteditor und Konsole) gearbeitet. Die Grundkenntnisse werden über Theorieeinheiten und Übungen vermittelt und Abläufe werden meist mit textbasierter Ausgabe repräsentiert. Das Gezeigte zu reproduzie-

ren, zu verstehen und anzuwenden ist der Kernpunkt bei der Kompetenzgewinnung. Dies ist bei begrenzter Unterrichtszeit und ungeeigneten Methoden der Wissensvermittlung eine besondere Herausforderung.

1.1 Problemstellung

In Oberstufen an Gymnasien an Schulen Österreichs ist die Zeit, welche pro Jahr für Programmieren eingesetzt werden kann, aufgrund des umfangreichen Lehrplans auf nur acht Wochen pro Jahr beschränkt. Aufgrund der mangelnden Zeit bekommen die SchülerInnen meist einen Theorieüberfluss zu hören und müssen im Selbststudium mit Übungen Programmieren lernen. In der Praxis geschieht dies meist so, dass ein oder zwei engagierte SchülerInnen die Übungen vorprogrammieren und der Rest der Klasse die Vorlage abtippt.

Es gilt also, den SchülerInnen im Unterricht schnell, effektiv und nachhaltig Wissen zu vermitteln und Kompetenz aufzubauen. Da aktives Lernen im Unterricht beim Programmieren mehr oder weniger durch Theorie, spielerisches Simulieren von Programmabläufen, teilweise kooperatives Arbeiten und manuelles Tippen von Code bereits vorkommt, wird hier der Fokus auf die meist vernachlässigte, visuelle Repräsentation von Programmabläufen und Code zur Steigerung der Effektivität gelegt. Es wird durch gezielte Verwendung von Computergrafik eine Methode entwickelt, welche das Verstehen, Reproduzieren und Anwenden beim Schüler, bei der Schülerin durch visuelle Darstellung beschleunigt.

Die wissenschaftlich überprüften Methoden von visuellem Lernen in der Programmierung sind Basis für die Entwicklung der im Anschluss vorgestellten Methoden für Oberstufen an Gymnasien in Österreich. Sie beschäftigen sich mit einer effektiven, visuellen Repräsentation von geschriebenem Code zum besseren Verständnis der gewünschten Lehrinhalte.

1.2 Motivation

Als Lehrer bin ich seit einigen Jahren auf der Suche nach einer effektiven und schnellen Methode zur Vermittlung von Fähigkeiten und Wissen im Bereich der Programmierung. Zu Beginn habe ich mich an gängige Literatur und Lehrbücher gehalten, welche als Vorlage für meinen Unterricht dienten. Im Bereich der Programmierung war es schwierig, SchülerInnen die Denkweise eines Programmierers zu vermitteln, da der Lehrinhalt für SchülerInnen aufgrund der textuell basierten Ausgabe schwierig zu fassen war und die Programmierübungen auf die SchülerInnen trocken wirkten.

Die Begriffe und die praktischen Übungen schienen für die SchülerInnen sehr abstrakt und ungreifbar zu sein. Auf der Suche nach alternativen Möglichkeiten zur attraktiveren Gestaltung des Unterrichts probierte ich mit Javascript und webbasiert zu programmieren. Ich merkte, dass SchülerInnen mit den praktischen Beispielen leichter umgehen konnten und mehr verstanden. Als Steigerung suchte ich mir eine Javascript Bibliothek zur grafischen Programmierung im canvas-HTML Element. Diese erste Bibliothek war PaperJS. Sie lieferte eine einfache Möglichkeit, grafisch im Browser zu programmieren. Die SchülerInnen waren motiviert, da die Visualisierungen besser verständlich waren und sie in Richtung Computerspielprogrammierung arbeiten konnten. Weiter merkte ich, dass der Unterricht leichter von der Hand ging, da die SchülerInnen schneller begriffen und eine Zeitersparnis durch eine effektiv zu scheinende Lernkurve deutlich merkbar war.

Dies brachte mich auf den Gedanken, dies wissenschaftlich zu erforschen und zwar stellte ich mir die Frage, ob ich SchülerInnen in der Oberstufe mit geringen Computerkenntnissen effektiv und schnell die Denkweise eines Programmierers vermitteln kann, wenn die Ausgabe grafisch basiert ist, im Gegensatz zur textbasierten Ausgabe. Mit diesem Ziel vor Augen begann ich, unterschiedliche, grafisch basierende Computerspiele mit den SchülerInnen zu programmieren und den SchülerInnen die Denkweise eines Programmierers zu vermitteln. Mich reizte es, diese Erkenntnisse zu erforschen und wissenschaftlich zu belegen. So entstand der Grundgedanke zu dieser Arbeit.

1.3 Zielsetzung

Ziel dieser Arbeit ist es, Möglichkeiten zur Steigerung der Lerneffizienz beim Erlernen des Programmierens aufzuzeigen. Eine geschickte Abänderung bei der Ausgabe von ausgeführtem Code und grafischer Repräsentation des Ergebnisses soll die Aufnahme von Wissen und die Fertigkeiten des Programmierens beschleunigen. Die menschliche Eigenschaft, sich audiovisuelle Repräsentation besser zu merken, soll die Lerneffizienz steigern. Dies wird durch zwei Testgruppen geprüft, welche zum einen textbasiertes Programmieren mit textueller Ausgabe durchführen und zum anderen visuell basiertes Programmieren mit grafischer Ausgabe erlernen. Eine Überprüfung der gewonnenen theoretischen und praktischen Fertigkeiten am Ende der Unterrichtssequenz zeigt das Ergebnis. Die Interpretation der Ergebnisse soll Erkenntnis über die Unterrichtsmethode mit grafisch basierter Ausgabe bringen.

Mit einer wöchentlichen Einheit von zweimal 50 Minuten auf acht Wochen verteilt wurde die Unterrichtssequenz angepasst auf den Lehrplan ausgelegt. Die SchülerInnen steigen

mit Grundkenntnissen im Umgang mit dem Computer in das Programmieren ein, und am Ende aller Einheiten sind die Anforderungen des Lehrplans zum Programmieren erfüllt.

1.4 Aufbau der Arbeit

Das erste Kapitel dieser Arbeit widmet sich den Grundlagen. Die aktuellen Forschungen im Bereich des Programmierenlernens mit grafischer Unterstützung werden in diesem Abschnitt beschrieben. Die gewonnen Erkenntnisse an Universitäten und Schulen, die Einflussfaktoren und nötigen Grundkenntnisse für das Programmieren werden für die Erstellung des Unterrichts in Kapitel 3 benötigt. Die Lernmethode „Aktives Lernen“, welche heutzutage als Basis für den aktuellen Unterricht in Form von Flipped Classroom oder Blended Learning dient, wird zu Beginn des Kapitel 2 dem Leser nähergebracht. Die Bedeutung der Visualisierung in der Wissensvermittlung wird im Anschluss erklärt und die Softwarevisualisierung beim Programmieren im Zusammenhang mit verschiedenen Programmierwerkzeugen wird vorgestellt. Die Programmierwerkzeuge dienen dem Zweck, geschriebenen Code auf unterschiedliche Art und Weise zu visualisieren, was dem Sinn dieser Arbeit entspricht und deshalb erwähnt wird. Die Voraussetzungen für den Unterricht werden durch eine Analyse des aktuellen und geplanten Lehrplans beschrieben und der schrittweise Aufbau einer Unterrichtseinheit, welcher zum Verständnis dient, wird in Kapitel 2 ebenfalls erklärt. Die eingesetzten Entwicklerwerkzeuge zur Programmierung in den Lehreinheiten sind Notepad++, Chrome und Javascript beziehungsweise Processing (p5.js). Diese werden näher im zweiten Kapitel betrachtet.
Das dritte Kapitel beschreibt die Unterrichtssequenzen, welche in den Testgruppen unterrichtet wurden. Es werden die Lehrinhalte, Beispiele, Kontrollfragen und Übungen jeder Einheit gezeigt. Zu Beginn wird ein Einstiegstest durchgeführt, welcher den Wissensstand und die Fertigkeiten der SchülerInnen am Anfang der Unterrichtssequenz ermittelt. Am Ende des Kapitels werden die abschließenden praktischen Aufgaben und Präsentationen der SchülerInnen beschrieben.
Im Kapitel 4 werden die abschließende Überprüfung der erlernten Programmierkenntnisse aus den Unterrichtssequenzen und die Denkweisen eines Programmierers überprüft. Die Analyse der erzielten Ergebnisse wird durchgeführt und anschließend interpretiert. Am Ende steht eine zusammenfassende Stellungnahme und ein Ausblick in die Zukunft.

2 Grundlagen

In diesem Kapitel werden die Grundlagen für die vorgestellte Unterrichtsmethode beschrieben. Das Ziel ist es, die Grundlagen und Begriffe so zu vermitteln, dass auch ein Techniker ohne pädagogischen Hintergrund versteht, worum es in dieser Arbeit geht. Der Zusammenhang zwischen „Aktivem Lernen“ und der visuell unterstützten Ausgabe beim Programmieren wird in diesem Kapitel vorbereitet. Die Bedeutung der Visualisierung beim Lernen soll hervorgehoben werden. Es wird auf die Software- und Algorithmusvisualisierung im Zusammenhang mit Programmieren lernen eingegangen. Die zu vermittelnden Lehrinhalte im Wahlpflichtfach Informatik an der Oberstufe der Gymnasien Österreichs werden für das folgende Kapitel beschrieben. Die Unterrichtseinheiten decken den Lehrinhalt für das Programmierenlernen ab. Am Ende des Kapitels werden aktuelle Forschungen in dem Bereich „visuell unterstütztes Programmieren lernen“ analysiert und eine Unterrichtseinheit wird skizziert.

2.1 Programmieren an Universitäten und Schulen

Das Thema, wie ein Programmierneuling effektiv programmieren erlernt, wird in verschiedenen “Educational Tracks” auf unterschiedlichen Konferenzen behandelt [1, 7, 8]. Motivation dafür ist die Herausforderung, einem Anfänger die logisch-algorithmische Denkweise eines Programmierers nahezubringen. Programmieren besteht aus mehreren Fähigkeiten, welche beim Lernenden ausgebildet werden müssen, beginnend bei Regeln zur Syntax bis hin zur Semantik, dem Verstehen von Abläufen und Strukturen [11]. Unterschiedliche Ansätze vermitteln algorithmisches Denken und syntaktisch korrektes Programmieren unter Zuhilfenahme grafischer Mittel zur effektiven und nachhaltigen Vermittlung von Wissen [1, 5].

2.1.1 Programmieren

Der Prozess des Programmierens ist ein vielschichtiger und komplexer Prozess. Es werden analytische, mathematische und logische Fertigkeiten vorausgesetzt, um Spezifikationen zu lesen, in Algorithmen umzuwandeln und syntaktisch korrekten Code zu schreiben. Weiter müssen Grundkenntnisse am PC vorhanden sein, die den routinierten Umgang mit dem Computer gewährleisten. Das Abspeichern von Dateien, das Kompilieren und das Ausführen von Code muss möglich sein. Eine weitere Kompetenz erstreckt sich auf das

Testen von geschriebenem Code und das Auffinden von Fehlern. All die beschriebenen Fertigkeiten machen Programmieren aus [11].

2.1.2 Einflussfaktoren beim Programmierenlernen

Die Schwierigkeit des Programmierens liegt in den unterschiedlichen Fertigkeiten, die beim Programmieren benötigt werden (Problemlösung, Mathematik, Benutzung des Rechners…) Die herausfordernden Einflussfaktoren beim Erlernen des Programmierens sind [11]:

- Unterschiedliche Fähigkeiten
- Unterschiedliche Prozesse
- Programmiersprache
- Neuartigkeit
- Interesse
- Ruf des Programmierens
- Geschwindigkeit des Lernens

Die unterschiedlichen Fähigkeiten, die beim Programmieren benötigt werden, können hierarchisch betrachtet werden. Studenten lernen die Basis mit der korrekten Benutzung der Syntax und steigern ihre Fähigkeiten mit Semantik, Struktur und Programmierstil [11].

Die Entwicklung eines Programms basiert auf verschiedenen Prozessen. Diese Prozesse hängen mit der Spezifikation und der Weiterverarbeitung dieser zusammen. Es muss eine Spezifikation in einen Algorithmus umgewandelt werden, der wiederum in einen Programmcode übersetzt wird [11, 12]. Dieser Prozess ist für erfahrene Programmierer einfach, ein Programmierneuling stellt sich hier meist einer Herausforderung: aus den anfänglich erlernten syntaktischen Fähigkeiten einen Algorithmus zu entwickeln. Die Optimierung des Algorithmus ist wiederum ein anderes Thema.

Die Frage nach der besten Programmiersprache für einen Einsteiger sollte nicht von Relevanz sein, denn das zu Erlernende ist die Art und Weise des Programmierens. Allerdings sind manche Programmiersprachen für den optimierten Einsatz in der Industrie geschaffen und somit für das Unterrichten nicht geeignet, andere Programmiersprachen wurden für den Unterricht entwickelt. Die Programmiersprache ist das nötige Mittel, um Programmieren zu lernen [11].

Für viele Programmieranfänger ist die Thematik des Programmierens eine komplett neue Tätigkeit. Diese Neuartigkeit muss in eine gängige Vorgehensweise transformiert werden. Dadurch steigert der Lehrer die Sicherheit beim Lernenden, was eine Beschleunigung beim Lernen bewirkt [13].

Das Interesse als weiterer Einflussfaktor zielt auf die Aufmerksamkeit beim Erlernen des Programmierens und in weiterer Folge auf die Aufnahmefähigkeit ab. Die Programmierung von mathematischen Funktionen oder Statistiken oder die syntaktischen Einstiegsübungen können auf einen Programmierneuling abschreckend wirken und somit die Aufnahmeeffektivität verringern [11].

Der Ruf des Programmierens hat einen sehr theoretischen und schwierigen Charakter. Meist ist dieser Ruf von Vorgängerklassen und anderen Anfängern geprägt. Dazu kommt meist die Vorstellung, dass Programmierer ausschließlich „Nerds“ sind. Dieser Ruf hat Einfluss auf die Motivation und Einstellung anderer Programmieranfänger [11].

Die Geschwindigkeit des Lernens als Einfluss auf die Effektivität des Lernens ist mit dem Abschluss des Kurses nach einer bestimmten Zeit extern gesteuert. Dies beeinflusst den Lerner durch eine teils stressige Lernsituation. Die Komplexität des Programmierens würde eine individuellere Vorgehensweise für SchülerInnen fordern.

2.2 Aktives Lernen

Die gewählte Lernmethode beim Unterrichten von Inhalten beeinflusst den Lernerfolg bei Studierenden und SchülerInnen. Ein Ansatz, SchülerInnen effektiv und nachhaltig Wissen zu vermitteln, ist aktives Lernen [3, 6].
Aktives Lernen bedeutet, es werden unterschiedliche Techniken des Unterrichtens in den Lehreinheiten eingesetzt und der Lernende bekommt durch unterschiedliche audiovisuelle Reize ein besseres Verständnis des Inhalts und die Effektivität im Lernprozess wird gesteigert [3]. Zu den eingesetzten Techniken zählen manuelles Arbeiten, Problemlösen, kooperatives Arbeiten, Simulationen, Spiele und visuelles Lernen [3]. Aktives Lernen gehört zum Konstruktivismus, der das Lernen als eine Summe von Konstruktionsprozessen sinnesphysiologisch, neuronal, kognitiv sowie sozial beeinflusst betrachtet und gewonnene Kenntnisse in Lernprozesse und Gestaltung von Lernumgebungen ableitet.
Die Basis für aktives Lernen bietet die Lernpyramide audiovisueller Methoden [6].

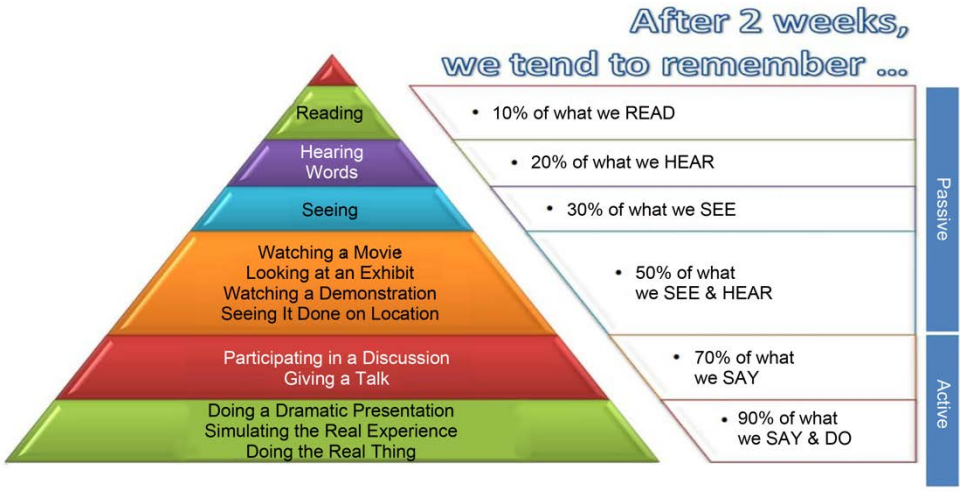

Abbildung 1: Audiovisuelle Methoden im Unterricht [30]

In der obigen Pyramide wird die Effektivität des aktiven Lernens dargestellt und die Wichtigkeit von Simulationen, interaktiven Arbeiten und visuellen Repräsentationen aufgezeigt. Die Pyramide beschreibt die eingesetzten Lehrmethoden und ihre Effektivität gemessen in der Erinnerung des Gelernten nach zwei Wochen. Es ist deutlich zu sehen, dass aktives Zutun während des Unterrichts am effektivsten ist. Die passiven Lehrmethoden in Form von Lesen, Hören und Sehen sind hinter der aktiven Lehrmethode angesiedelt, wobei Sehen und aktives Tun den höchsten Anteil und die beste Erinnerungsrate aufweisen.

2.2.1 Bedeutung von aktivem Lernen

Aktives Lernen wird oft in wissenschaftlichen Berichten erwähnt und hat im Bereich des Unterrichts eine wichtige Stellung. Aus diesem Grund soll die Bedeutung von aktivem Unterricht genauer untersucht werden.

2.2.2 Aktives Lernen im Unterricht

In den 1980er Jahren forderten einige führende Personen an höheren Bildungseinrichtungen die Einbindung der Studenten in den Prozess des Lernens. Bis dahin war die vorherrschende Methode des Unterrichts ausschließlich Frontalunterricht [3]. Allerdings hat die Umsetzung des aktiven Lernens einige Herausforderungen mit sich gebracht. Die Grundhaltung gegenüber aktivem Lernen ist positiv, oft wurde auch das Zuhören der Studierenden als aktiv bezeichnet. Was einen aktiven Unterricht ausmacht, ist allerdings mehr: Studierende sollten lesen, schreiben, diskutieren und in die Problemlösung involviert sein. Das aktive Lernen muss allerdings geführt und strukturiert sein [3]. Die Wissenschaft hat die besten Lernstile außerhalb einer klassischen Vorlesung nachgewiesen, diese müssten von den Lehrenden unterstützt werden.

Beispielsweise könnte während der Vorlesung auf Pausen geachtet werden, welche Studierende nutzen können, um ihre Mitschriften zu komplettieren. Darüber hinaus könnten kurze schriftliche Überprüfungen in der Stunde gemacht werden und die Ergebnisse könnten im Plenum diskutiert werden. Eine weitere Möglichkeit besteht darin, eine Feedback-Vorlesung zu machen, die aus zwei Minivorlesungen besteht, mit einer Gruppenbesprechung zwischen den Vorlesungen oder eine Vorlesung mit einer Dauer von 20-30 Minuten, in der die SchülerInnen zuhören und danach Aufzeichnungen oder Übungen absolvieren [3]. Die Größe der Klasse stellt eine Herausforderung für aktives Lernen dar, jedoch gibt es auch hier Vorstellungen von aktivem Lernen. Eine Frage könnte während der Vorlesung ausgearbeitet werden und mit dem/der SitznachbarIn rechts oder links ausgetauscht werden. Weitere Maßnahmen sind die Simulation, Problemlösungen für Case Studies oder aber auch visuelle Darstellung von Sachverhalten, um den Fokus auf das Wesentliche der Vorlesung zu lenken.

2.2.3 Herausforderungen bei aktivem Lernen

Eine Veränderung im Unterricht ist schwierig und stellt eine Herausforderung für Institutionen dar. Klassische Lernmethoden haben sich im Unterricht verankert, die Angst vor Veränderung ist vorhanden oder die Anreize für Veränderung im Unterricht sind gering. Der nächst Einflussfaktor für aktives Lernen ist Zeit: die Zeit für aktives Lernen ist meist in ausreichender Form gegeben, die Zeit für die Vorbereitung des Lehrenden jedoch fehlt, die Umsetzung von aktivem Lernen in großen Klassen erfordert Kreativität. Auch das Fehlen von geeigneten Unterrichtsmaterialien, Ausstattung und Ressourcen ist eine Erschwernis. Der größte Risikofaktor in der Veränderung des Unterrichts ist, dass SchülerInnen den neu gestalteten Unterricht nicht annehmen. Der Lehrende fühlt sich dadurch ohne Kontrolle und es wird Kritik am unorthodoxen Unterricht geübt [3].

2.3 Umsetzung von aktivem Lernen

Die Form des Unterrichts sollte zu Beginn langsam an aktives Lernen angepasst werden. Eine zu große Veränderung birgt hohe Risiken bei der Annahme des Unterrichts und der Sicherheit des Vortragenden mit sich [3]. Aktives Lernen hat seine Berechtigung, da es den Unterricht anregend gestaltet und durch verschiedene Reize das Gehirn der Studierenden anregt und die Aufnahme durch Interesse erhöht.

2.4 Vorteile von aktivem Lernen

Aktives Lernen beschäftigt sich damit, den Lernenden durch verschiedene Lernmethoden bestmöglich zu fördern. Visuelle Repräsentation von Lerninhalten oder auch Problemlö-

sungen in Form von Übungen gehören zum aktiven Lernen. Diese Tatsache unterstützt die Methode der Programmierung mit visueller Repräsentation, welche in dieser Diplomarbeit erklärt wird, sehr.

Beim aktiven Lernen merken sich die Studenten zwischen 70% und 90% der Lehrinhalte über zwei Wochen hinweg. Bei passiven Methoden wie Vorlesungen merken sich Studenten zwischen 10% und 30% [6]. Dadurch ist ein deutlicher Vorteil von aktivem Lernen gegenüber der passiven Methode gegeben [31, 32, 33]. Selbst Blended Learning kann als aktives Lernen betrachtet werden, unter der Prämisse, dass ausreichend Ausstattung dafür vorhanden ist, da es Anreize setzt und eine alternative Methode zum klassischen Vortrag darstellt [34]. Eine andere Variation von aktivem Lernen ist der "Flipped Classroom" [53]. Beim „Flipped Classroom" werden die Lehrinhalte nicht wie beim klassischen Unterricht durch Vortrag des Lehrers vermittelt, sondern die SchülerInnen erarbeiten die Lehrinhalte durch Videos oder Screencasts und wenden die Erkenntnisse auf Übungen im Unterricht an. Der Lehrer ist dabei Coach und kann individuell unterstützen.

Es ist wissenschaftlich erforscht, dass Studierende das Lernen von Programmieren schneller aufgeben, wenn sie starr die Programmiersprache lernen und keine Möglichkeit zur Interaktivität haben. Durch Kollaboration im Unterricht und Erarbeiten von praktischen Übungen unter Zuhilfenahme von Lehrmaterialien wird das Interesse am Programmieren gesteigert und die Lerneffizienz steigt [39]. Die benutzten Lernmaterialien sind visuell aufbereitet durch Animationen und Visualisierungen und die Messungen durch Tests am Ende des Kurses zeigten Verbesserungen in der "Flipped Classroom" Methode.

Ein weiterer Teil des aktiven Lernens sind Visualisierungen und audiovisuelle Repräsentation von Lehrinhalten. Mit Hilfe von Visualisierungen können Lernende einfacher Lehrinhalte erfassen und lernen dadurch schneller und effektiver [6]. Aus diesem Grund wird hier der Visualisierungsaspekt näher betrachtet. Speziell wird auf die Softwarevisualisierung eingegangen, um zu verstehen, was die Visualisierung im Bereich der Programmierung für Vorteile für den Lernenden hat.

2.5 Visualisierung

Zum aktiven Lernen gehört auch das Visualisieren von Lehrinhalten zur einfacheren Vermittlung von Wissen. Das Ziel der Visualisierung ist die Schaffung von Information durch das menschliche visuelle System im Gehirn mit der Zeichnung von Bildern auf den Computerbildschirm [24]. Die zentralen Punkte der Visualisierung sind physiologischer, psychologischer und technischer Natur.

Die Wahrnehmung erfolgt durch die menschliche Verarbeitung, Bewusstsein, Argumentation und Lernen. 75% aller Information der realen Welt werden visuell aufgenommen, 13% der Informationen werden durch das Hören aufgenommen und 12% werden durch restliche Sinne erfasst [24].

Das menschliche Gehirn kombiniert diese Informationen, es besteht aus zwei Gehirnhälften:

- die linke Hälfte, ist für Verbales, Analytisches, Rationales und Sequentielles zuständig
- die rechte Hälfte ist für Nonverbales, Synthetisches, Intuitives und Paralleles zuständig

Die Visualisierung hilft, beide menschliche Gehirnhälften zu entwickeln. Für Visualisierungen spricht auch die Möglichkeit, sich Bilder besser merken zu können als gelesenen Text [24]. In einem Test wurden Bild- und Texterinnerungen abgeprüft und das Ergebnis spricht für die Visualisierungen mit 1,5% Fehlerrate, im Gegensatz zu Texten mit 11,8%. Das Ziel von Informationsvisualisierung ist die Anzeige von abstrakten Zusammenhängen und Beziehungen auf natürliche Art und Weise zur Unterstützung des Betrachters beim Verständnis [24].

Die Softwarevisualisierung ist entweder ein Teil der Software zur Visualisierung von Daten oder sie beschäftigt sich mit der Visualisierung von Code und Zusammenhängen wie Datenbankstrukturen usw. während des Entwicklungsprozesses und dient dort als Repräsentation, Navigation oder Analyse [21].

Softwarevisualisierung

Die Softwarevisualisierung dient als Inspiration für die entwickelten Lehreinheiten im nächsten Kapitel und ist die Inspiration für die unten entwickelte Unterrichtssequenz. Weiter ist das Verständnis von Softwarevisualisierung für die in diesem Kapitel beschriebenen wissenschaftlichen Artikel erforderlich.

Eine Definition von Softwarevisualisierung kann sein [23]:
"… is the visualization of artifacts related to software and its development process."

Eine Definition von Softwarevisualisierung ist auch [1]:
"Software visualization is the use of the crafts of typography, graphic design, animation and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software" [22].

Der Prozess der Softwarevisualisierung beinhaltet die Visualisierung von

- Struktur
- Verhalten
- Evolution der Software

Die Struktur in der Software besteht aus statischen Komponenten und Beziehungen zwischen den Komponenten, welche ermittelt werden können, ohne das Programm selbst auszuführen. Dazu zählen der Programmcode, die Datenstrukturen, Aufrufe und vieles mehr. Das Verhalten der Software bezieht sich auf die Ausführung des Programms mit realen und abstrakten Daten. Der Ablauf des Programms wird als Zustandsautomat gesehen. Entweder rufen Funktionen andere Funktionen auf oder Objekte arbeiten mit Objekten (abhängig von der Programmiersprache). Die Evolution von Software kann beim Ausbessern oder Erweitern von Code als ständiger Erweiterungsprozess gesehen werden.
Diese drei Teilbereiche fließen in die Softwarevisualisierung ein. Das Verstehen eines Programmes besteht aus Analyse, Abstraktion und Generalisierung [25]. Softwarevisualisierung dient zum Verstehen von Softwaresystemen zur Steigerung der Produktivität und des Softwareentwicklungsprozesses.

Die Erkenntnis aus der Softwarevisualisierung wird in dieser Arbeit genutzt, um Unterrichtseinheiten zu entwickeln, welche Programmieren lernen mit Hilfe von grafischen Elementen ermöglicht. Die grafische Ausgabe von einfachen Algorithmen und Zusammenhängen beim Programmieren stehen in dieser Arbeit im Fokus.

2.6 Theoretische Aufarbeitung derzeitiger Forschungen und Related Work

Die Erkenntnis der effektiveren Wissensvermittlung durch Animation und Visualisierung im Bereich des Programmierens wurde an Universitäten aufgegriffen und wissenschaftliche Forschungen beschäftigen sich mit diesem Thema.

2.6.1 Programmieren an Universitäten mit grafischer Unterstützung

Diese Forschungen sind grundsätzlich ähnlich, denn durch geschicktes Kombinieren von trockener Theorie und grafischen Elementen wird das Lernen erleichtert und sowohl das Verstehen der Syntax und Semantik, als auch der Algorithmik verbessert [7]. An Universitäten ist diese Methodik der Wissensvermittlung bereits öfters eingesetzt worden [7,8], mit der Herausforderung der Schaffung eines geeigneten Umfeldes für die Studierenden [9, 10]. Hier mussten eigens programmierte Entwicklungsumgebungen geschaffen werden, in denen die Übungen der Studierenden abgehalten wurden [3, 8, 10]. Der Aufwand,

eine grafisch basierte Methodik zur Vermittlung von Programmierkenntnissen zu verwenden, hat sich allerdings bewährt, denn die Ergebnisse der Studierenden bei Überprüfungen waren um einen Grad besser als bei nicht grafischen Methoden zum Erlernen von Programmieren [1]. Somit wird der Vorteil einer Visualisierung beim Begreifen der Programmstrukturen und Verstehen der Programme bei Programmiereinsteigern gezeigt [1, 10].

Im wissenschaftlichen Artikel „Effectiveness of Program Visualization in Learning Java: a Case Study with Jeliot 3" wird die Effektivität der visuellen Komponente beim Programmieren als Unterstützung zum Verstehen von Programmierinhalten deutlich herausgearbeitet. Mit der Hilfe von 400 Studierenden wurde eine statistisch eindeutige Verbesserung bei Studenten mit stark unterstützter, grafischer Repräsentation beim Programmieren festgestellt [1].

Ein anderes Forschungsteam beschäftigte sich im Artikel „Using blended learning to improve student success rates in learning to program" mit einer Mischung der Unterrichtstechniken und setzte zum klassischen Frontalunterricht zusätzlich Blended-Learning ein. Durch die Veränderung des Unterrichts und die zusätzliche Möglichkeit online zu lernen, wurden deutlich messbare Verbesserungen bei den Prüfungsergebnissen der Studierenden erzielt [14].

Der Artikel „What a Novice Wants: Students Using Program Visualization in Distance Programming Course" beschreibt, wie Neulinge in der Programmierung mit der Codevisualisierung und Ablaufsimulation Jeliot 3 umgehen und durch besseres Verständnis mit Hilfe der Visualisierung des Codes und Simulation des Ablaufs Lernerfolge erzielen [15]. Das Verständnis der Studierenden stieg und Programmieren konnte schneller erlernt werden.

In der Studie „A Meta-Study of Algorithm Visualization Effectiveness" wurden unterschiedliche Experimente zum Einsatz von Visualisierung in der Programmierung durchgeführt. Abgesehen von den Schwierigkeiten, Lehrende zu überzeugen, neue Unterrichtsmethoden in ihrem Unterricht einzusetzen, war das Ergebnis für Visualisierung in der Programmierung eindeutig positiv. Die Ablaufvisualisierung wurde als Hilfe von den Studenten angenommen und das Verständnis in der Programmierung erhöht [16].

Mit Hilfe der Forschung „Methodologies for studies of program visualization" wurden Möglichkeiten zum Studieren von menschlichem Verhalten, Interaktion und Lernverhalten von Programmvisualisierungen dargestellt: Klassenzimmerstudien, kontrollierte Experi-

mente für Interaktionen und kognitive Wahrnehmung, Fragebögen zur Sammlung von Daten für Antworten, mögliche Hypothesen und Gewinnung von Benutzerproblemen [17].

Eine weitere Arbeit beschäftigt sich mit der logischen und syntaktischen Überprüfung von Code. Es wurde in diesem Projekt ein Überprüfungstool eingesetzt, welches visuelles Feedback über den programmierten Code gab, genannt der “Verificator”, und die Leistung der Studenten durch visuelles Feedback steigerte und zu erhöhtem Lernerfolg führte [20].

Eine weitere Methode, „Teaching Programming in the Higher Education not for Engineering Students”, beschäftigt sich mit einer spielbasierten Programmierung und Visualisierung von Code [7]. Dieser Ansatz ist für den Unterricht interessant, da die Studenten mit visualisiertem Code in Form eines Computerspiels leichter zu programmieren gelernt haben als Studierende, die auf rein textueller Basis programmierten [7].

2.6.2 Programmieren an Schulen

An Oberstufen der Gymnasien an Österreichs Schulen gibt es im Wahlpflichtfach Informatik wenig Zeit, Grundkenntnisse des Programmierens zu vermitteln. Diese Grundkenntnisse richten sich nach Lehrplänen mit vorgeschriebenen Inhalten. Für Lehrer stellt dies eine Herausforderung dar. Es sollen die Basiskenntnisse an SchülerInnen in sehr kurzer Zeit mit bester Qualität vermittelt werden. Effektiv kann in etwa mit zwei Monaten (acht Lehreinheiten) pro Jahr gerechnet werden, in denen ausschließlich Programmieren unterrichtet werden kann. Dies resultiert daraus, dass es verschiedenste Bereiche im Wahlpflichtfach wie Datenbanken, Informatik und Gesellschaft, IT-Security und vieles mehr gibt, die auch durchgenommen werden müssen. Deshalb ist es für Lehrer wichtig, den SchülerInnen die Grundlagen im Programmieren effizient, schnell und bestmöglich beizubringen.

Um das zu erreichen, sind einige Faktoren von entscheidender Relevanz: erstens, eine Programmiersprache mit geringer Einarbeitungszeit, die ohne viel Aufwand auf jedem System ausgeführt werden kann und möglichst alle Konzepte und Bereiche im Programmieren für SchülerInnen abdeckt. Ist eine passende Programmiersprache gefunden, wird am Verständnis für SchülerInnen gearbeitet. Algorithmisches Denken, das Verstehen von Zusammenhängen und die nötigen Programmierkenntnisse sollen den SchülerInnen innerhalb des Wahlpflichtfaches Informatik vermittelt werden. Das Schreiben von Code in einem Texteditor und das Überprüfen der Ausgabe in einem Terminalfenster sind derzeitige Ansätze, wie Programmieren vorwiegend an Bildungseinrichtungen vermittelt wird.

Recht rasch wirkt dies für einen Neuling im Programmieren als “trockene” Tätigkeit, die Aufmerksamkeit sinkt und somit auch das Verständnis. Daraus resultiert, dass die SchülerInnen mehr Zeit braucht, um den Lernstoff zu begreifen und die Effektivität sinkt.

2.6.3 Forschung zu grafisch unterstütztem Unterricht an Schulen

In der Arbeit „Learning programming from Scratch” werden SchülerInnen in der 5. Schulstufe mit Programmierung konfrontiert. Es wird verglichen, ob textuell basiertes Programmieren mittels Python oder der Einsatz von Scratch optimaler für den Lernerfolg ist. Im Ergebnis dieser Arbeit wird Scratch als geeignetes Tool dargestellt, um die breite Masse an Studierenden zu erreichen. Es wird erklärt, dass einige SchülerInnen keine Probleme haben, textbasiert zu programmieren und es zu verstehen. Mindestens zwei Drittel der SchülerInnen jedoch lernen visuell basiert besser [19].

Die wissenschaftliche Abhandlung „Learning with interactive computer graphics in the undergraduate neuroscience classroom” beschreibt die grafische Repräsentation von Lehrinhalten mittels Computerunterstützung [35]. Der Unterricht war für Klassen unter der Maturareife vorgesehen. Durch die Arbeit mit den grafischen Elementen werden der Lernzyklus und die Lerneffektivität deutlich unterstützt. Die Überprüfung des Gelernten bei den Studierenden zeigte positive Ergebnisse. Außerdem wird in der Studie ein klares Interesse für die grafische Variante bekundet.

Die Studie „Work in Progress - Impact of Graphical Programming Environments on Learning and Understanding Programming Concepts” fokussiert sich auf das Interesse der Studierenden beim Schreiben von textbasiertem Code. Technisch interessierte Personen tendieren dazu, visuell zu lernen, basierend auf dem Modell „FelderSilverman Learning Styles Model” [41]. Die traditionell textbasierten Programmiersprachen entsprechen nicht dem Lernstil, den die Studierenden und SchülerInnen haben – visuelle Lerner. Mit einer Mischung aus visueller Repräsentation und textbasierter Programmierung wird dem Lerntypus visueller Lerner eher entsprochen [41].

2.6.4 Literatur zu grafisch unterstütztem Programmieren lernen

An dieser Stelle ist das Buch von Oliver Deussen und Thomas Ningelgen zu nennen: Programmieren lernen mit Computergrafik [58]. Dieses beschäftigt sich mit der Vermittlung von Programmierkenntnissen unter zuhilfenahme von Computergrafik und der Programmiersprache Java sowie der Bibliothek Processing. Der Aufbau des Buches führt den Leser durch geeignetes Einsetzen von Visualisierungen in die Denkweise des Schreibens von Algorithmen und Code ein. [58] Der Fokus des Buches liegt auf der raschen Vermittlung

von Programmierkenntnissen ohne explizit auf den Vorteil der grafischen Visualisierung für den Lernenden einzugehen. Die Autoren strukturieren die Vermittlung von Programmierkenntnissen in folgende aufeinander basierende Bereiche [58]:

- Einstieg
- Variablen, Ausdrücke, Typen und Funktionen
- Kontrollstrukturen
- Arrays
- Prozeduren
- Algorithmen
- Objektorientierung

Zu Beginn fokussieren sich die Autoren des Buches auf die Installation beziehungsweise die Ausführung von Processing. Danach werden Programmzustände und Algorithmen erklärt, um Verständnis für den Lernenden zu schaffen. Dabei werden Formen kreiert und Processing wird zur Programmierung und Visualisierung eingesetzt. Nun werden einfache Datentypen beschrieben und Variablen zum Schreiben von Code eingesetzt. Ausdrücke, Zeichenketten (Strings), Arrays sowie Operatoren und logische Ausdrücke werden gezeigt. Auch Funktionen als Teil der Programmiersprache werden beschrieben und der Übergang zu Kontrollstrukturen erfolgt. An dieser Stelle werden Blöcke, if-else-Anweisungen, for-Schleifen und while-Schleifen in Beispielen vermittelt. Der Einsatz von eindimensionalen und mehrdimensionalen Arrays sowie das Schreiben von Algorithmen und Prozeduren werden im Anschluss erklärt. Der letzte große Schritt ist die Einführung der Objektorientierung mit Klassen und Vererbung. Attribute und Methoden werden erläutert, Objekte und Klassen sowie Vererbung werden skizziert und Beispiele helfen der lernenden Person die zuvor vermittelten Inhalte zu verstehen. Grundsätzlich sind die Kapitel und Übungen des Buches sehr visuell fokussiert.

2.7 Kompetenzen nach dem Lehrplan

Der Lehrplan für Informatik wird an dieser Stelle analysiert. Er dient als Basis für die im nächsten Kapitel vorgestellten Unterrichtseinheiten zum Erlernen des Programmierens. Im Lehrplan für das Wahlpflichtfach Informatik aus dem Jahr 2004 sind folgende Aufgabengebiete festgehalten [36]:

- Grundprinzipien der Informationsverarbeitung
- Konzepte von Betriebssystemen- Aufbau und Funktionsweise von Netzwerken
- Datenbanken

- Lern- und Arbeitsorganisation
- Konzepte von Programmiersprachen
- künstliche Intelligenz
- Erweiterung der theoretischen und technischen Grundlagen der Informatik
- grundlegende Algorithmen und Datenstrukturen
- Informatik, Gesellschaft und Arbeitswelt
- Rechtsfragen

Zwei Bereiche beschäftigen sich mit Programmieren: Konzepte von Programmiersprachen und grundlegende Algorithmen und Datenstrukturen. Bei einer Gleichverteilung der verfügbaren Zeit innerhalb von drei Jahren bleiben für das Thema rund um das Programmieren in etwa vier bis sechs Wochen effektiv pro Jahr Zeit. Hierbei wurden die erste und letzte Woche als auch das letzte verkürzte Jahr in den 8. Klassen berücksichtigt. Für die Vermittlung der Grundlagen des Programmierens bleiben in etwa acht bis zwölf Wochen. Weitere Zeit im Bereich Programmieren muss für Modellierung, Softwaretesten und Ähnliches aufgewendet werden. Laut dem Vorschlag für den neuen Lehrplan im Wahlpflichtfach Informatik sind folgende Kompetenzen zu erlernen [37]:

- Wissen und Verstehen
- Anwenden und Gestalten
- Reflektieren und Bewerten

Diese Kompetenzen werden bei der Reifeprüfung durch Reproduktion, Transfer, Reflexion und Problemlösung abgefragt. Die Themengebiete bzw. Lehrinhalte werden an der Oberstufe an Gymnasien unterrichtet und konzentrieren sich unter anderem auf praktische Informatikkenntnisse [37]. Im folgenden Teil wird eine Auflistung der zu vermittelnden Lehrinhalte gegeben, wobei die für die in dieser Arbeit relevanten Lehrinhalte fett hervorgehoben sind [37]:

- Algorithmusbegriff erklären können
- Grundlegende Aufgaben und Problemstellungen algorithmisch und formalsprachlich in geeigneten Datenstrukturen beschreiben können
- Grundlegende Algorithmen entwerfen, diese formal darstellen, implementieren und testen können
- Aufgaben mit Mitteln der Informatik modellieren können
- Komplexere Algorithmen entwerfen, diese formal darstellen, implementieren und testen können (Erweiterung, Vertiefung)
- Aspekte der Prozeduralen, Funktionalen und Objektorientierten Programmierung nennen und an Beispielen erläutern können

- Aufgaben mit Mitteln der Informatik modellieren können. Vielfältige Algorithmen entwerfen, diese formal darstellen, implementieren und testen können (Erweiterung, Vertiefung)
- Den Begriff Datenbanken und andere in diesem Kontext wichtige Fachbegriffe beschreiben und an Beispielen erklären können
- Datenbankmodelle, Tabellen und ihre Beziehungsmuster sowie weitere Datenbankobjekte erklären können
- Daten strukturiert (in Tabellen) erfassen, abfragen, auswerten sowie Datenbanken modellieren und einfache automatisierte Datenbanklösungen entwickeln können
- Datenmodelle hinsichtlich der Datentypen, Redundanz, Integrität und Relevanz bewerten können
- Wesentliche informatische Konzepte und fundamentale Ideen der Informatik benennen und an Hand von Beispielen erklären können
- Bei der Lösung konkreter Aufgaben Heuristiken, Grundprinzipien und Konzepte der Informatik anwenden und informatische Modelle gestalten können
- Vielfältige Aufgaben mit Mitteln der Informatik modellieren können
- Vielfältige Algorithmen entwerfen, diese formal darstellen, implementieren und testen können
- Unterschiedliche Lösungsansätze in Bezug auf zugrunde liegende Konzepte reflektieren und in konkreten Handlungssituationen bewerten können
- Die Effizienz von Algorithmen bewerten können
- Unterschiedliche Lösungsansätze in Bezug auf zugrunde liegende Konzepte reflektieren und in konkreten Handlungssituationen bewerten können
- Wesentliche Aspekte und Methoden der Softwareentwicklung und des Softwareprojektmanagements erklären können
- Ein Softwareprojekt planen und durchführen können
- Die Schritte der Softwareentwicklung reflektieren können
- Die Angemessenheit der Entwicklungswerkzeuge grob einschätzen können
- Die Effizienz von Algorithmen bewerten können
- Gezielt nach Programmfehlern suchen und diese korrigieren können
- Bereiche beschreiben können, in denen sich Informatiksysteme bzw. Computer intelligent verhalten
- Den Unterschied zwischen menschlicher und maschineller Intelligenz erklären können, Merkmale menschlicher Intelligenz und künstlicher Intelligenz vergleichen und einschätzen können.
- Intelligente Informatiksysteme anwenden können

Die Lehrinhalte und Anforderungen an einen Absolventen des Wahlpflichtfachs Informatik sind im Fokus der Wissensvermittlung im Unterricht und werden für die praktische und

theoretische Testung am Ende der Unterrichtssequenz zur Ermittlung der Effektivität der grafisch basierten Ausgabe beim Erlernen des Programmierens herangezogen.

Abschließend ist zu erwähnen, dass die Anforderungen des zurzeit gültigen Lehrplans und die Anforderungen des Entwurfs des zukünftigen Lehrplans hinsichtlich der Programmierkenntnisse kontrolliert und in den Lehrinhalt des unten vorgestellten Unterrichts miteinbezogen sind.

2.8 Methode zur Steigerung der Effektivität des Unterrichts zur Vermittlung von Programmieren

In den vorigen Kapiteln ist die Wichtigkeit der visuellen Repräsentation deutlich geworden. Sie wird im aktiven Unterricht eingesetzt, dessen Ausprägung in den letzten Jahren in unterschiedlicher Form im pädagogischen Bereich an Universitäten und Schulen Einzug gehalten hat. Die visuelle Repräsentation von Inhalten ist aus unterschiedlicher Hinsicht wichtig: der Lernende benutzt beim Ansehen einer Visualisierung beide Gehirnhälften und 70% - 90% des Gesehenen wird sich vom Lernenden gemerkt, im Gegensatz zu 10% bei textuellen Inhalten.

Die hier vorgestellte Methodik zum effektiven Lernen von Programmieren berücksichtigt diese Erkenntnisse. Für eine signifikante Aussage über die Effektivität der neuen, angewandten Methode mussten zwei Unterrichtssequenzen modelliert werden und zwei zufällig gewählte, gemischte Testgruppen, SchülerInnen, mit ähnlichen Voraussetzungen zum Programmieren mussten gewählt werden. Mit einem Vortest zu Beginn, welcher die Fähigkeiten der einzelnen SchülerInnen im Bereich Mathematik und Logik, Kompetenz am Computer sowie Lösungskompetenz ermittelte, wurden die tatsächlichen Kenntnisse der Schülerinnen festgestellt. Beide Sequenzen setzen sich zum Ziel, Programmieren innerhalb von acht Wochen in zwei Unterrichtsstunden pro Woche den SchülerInnen beizubringen, mit dem Unterschied, dass die eine Gruppe textuell programmiert und die andere Gruppe grafisch unterstützt programmiert. Die gewählte Programmiersprache ist für beide Gruppen die gleiche. Die Repräsentation des programmierten Codes findet bei der einen Gruppe in Form eines textuellen Outputs statt, bei der anderen Gruppe wird ein visueller Output verlangt.

Mit Hilfe der Computergrafik und der Programmiersprache Javascript wird in dieser Arbeit gezeigt werden, dass eine geschickte Kombination aus Code schreiben und Grafik das Verständnis bei SchülerInnen optimal fördert und Kompetenzen vermittelt.

2.8.1 Programmiersprache

Das Lernen einer Programmiersprache stellt für viele Studierende und SchülerInnen eine Herausforderung dar. Die Schwierigkeit besteht in der Wahl der geeigneten Programmiersprache und darin, das Programmieren als Tätigkeit selbst zu verstehen [38]. Die gängigen Programmiersprachen sind Java und C oder C++. Leider sind diese Programmiersprachen sehr komplex. Diese Komplexität schreckt Studierende ab und vermindert die Aufmerksamkeit und Motivation, die wiederum das Lernverhalten und die Lerneffektivität beeinflusst. Für Studenten scheint es oft, dass Programmieren lernen das Lernen der Programmiersprache und ihrer Konstrukte ist (Kontrollfluss, Variablendefinition, und Ähnliches) [38]. Diese für Anfänger sehr abstrakten Zusammenhänge in einer Programmiersprache sind schwer zu begreifen und demotivieren SchülerInnen beim Lernen des Programmierens. Es gibt eine Tendenz, dass das Lernen des Programmierens mit der Wahl der Programmiersprache zusammenhängt. [38]. In der wissenschaftlichen Abhandlung „Gradually Learning Programming Supported by a Growable Programming Language" wird die Wichtigkeit der Vermittlung der Denkweise des Programmierens hervorgehoben, die Art und Weise wie Probleme gelöst werden, ohne Rücksicht auf die Programmiersprache. Die Wahl der Programmiersprache beim Programmieren steht im Hintergrund.
Aus diesem Grund wurde in der vorgestellten Methode eine Programmiersprache gewählt, die verschiedene Aspekte berücksichtigt. Diese sind die Anforderungen an den Unterricht, die Einfachheit der Umsetzung im Unterricht und die Möglichkeit der Flexibilität.

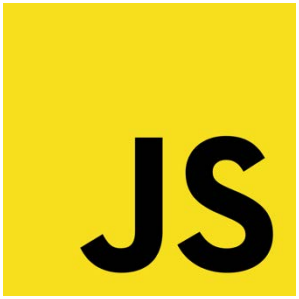

Abbildung 2: Javascript

In der Programmiersprache Javascript, das Logo ist oben gezeigt, sind folgende Konzepte der Programmierung möglich:

- Prozedurales Programmieren
- Objektorientiertes Programmieren
- Funktionales Programmieren

Das bedeutet, dass mit der Programmiersprache Javascript unterschiedliche Konzepte des Programmierens gelehrt werden können und der Fokus des Lernenden nicht auf dem Er-

lernen der Programmiersprache liegt, sondern auf dem Erfassen der Programmierkonzepte dahinter. Schlussendlich ist das Ziel das Verstehen, was Programmieren eigentlich ist.
Durch die Ausführbarkeit der Programmiersprache im Browser und im Backend ist Javascript auch für die Programmierung von Server-Client Architekturen geeignet, Socketprogrammierung oder Datenanbindungen können mit Javascript programmiert werden. Somit kann Javascript auch für diese Anwendungsgebiete in der Softwarearchitektur geeignet sein.

Ein weiterer wesentlicher Vorteil der Programmiersprache Javascript ist die Einfachheit der „Inbetriebnahme", wenn kein Backend in Javascript programmiert werden soll. Für Java oder C++ müssen meist Compiler oder Software Development Kits installiert werden. Das bedeutet eine Vorbereitung des Computers vor der ersten Programmierung. Weiter werden meist Integrated Development Environments eingesetzt. Diese erfordern ebenfalls eine Vorabinstallation vor dem Kurs. Es handelt sich bei den Vorbereitungen zwar um einmalige Tätigkeiten, allerdings müssen diese gemacht werden. Bei Javascript reicht es, einen Texteditor, meist vorhanden im Betriebssystem, und einen Browser, ebenfalls vorinstalliert im Betriebssystem, zu haben und die Programmierung von Javascript Code ist möglich.

Javascript unterstützt folgende Funktionalitäten [43]:

- Datentypen (primitive und nicht primitive)
- Kontrollstrukturen (if-else, switch, while, do-while, for, for-in, for-of)
- Variablen (Scope von Variablen)
- Funktionen
- Objekte
- Private Eigenschaften (Closures)
- Vererbung (prototype)
- Fehlerbehandlung (try-catch)
- Sicherheit (Sandbox)
- Bibliotheken

Die oben genannten Funktionalitäten sind für den Einsatz als geeignete Programmiersprache für den Unterricht laut Lehrplan von hoher Bedeutung. Alle möglichen im Lehrinhalt vorkommenden Aspekte werden berücksichtigt, um eine Umstellung der Programmiersprache während des Erlernens zu vermeiden und die Denkweise eines Programmierers reibungslos zu vermitteln.

2.8.2 Processing

Für die grafische Darstellung des Programmierten ist eine Bibliothek zur Visualisierung beziehungsweise Animation nötig. Diese grafische Komponente muss zur gewählten Programmiersprache passen und dem Sinn der in dieser Arbeit vorgestellten Methodik entsprechen. Die Voraussetzungen sind also die Kompatibilität zu Javascript und der Einsatz der Grafikbibliothek im Ausbildungsbereich. Nach einer Recherche wurde Processing und die davon abgeleitete Bibliothek P5.js wegen der Javascript Kompatibilität gewählt.

Processing wurde entwickelt, um die Programmierung interaktiver Grafiken zu vereinfachen. Für Programmieranfänger oder nicht so geübte Programmierer ist es schwierig, Code in C, C++ oder Java zu schreiben. In vergangenen Programmiersprachen (Logo, Basic) war es einfach, Visualisierungen zu produzieren. Processing orientiert sich an dieser Idee, Visualisierungen so einfach zu gestalten wie es in Logo und Basic früher möglich war. Das Ziel von Processing ist es, Lehrern und Kunststudenten eine Möglichkeit zu bieten, einfach mit Grafik am Computer zu arbeiten. Der Fokus in Processing ist Interaktion und Grafik vor Datenstrukturen und Textausgabe [55].

Abbildung 3: Processing

Processing mit dem oben gezeigten Logo (Abbildung 3) wurde über mehrere Jahre hinweg im Unterricht getestet. Es gibt zahlreiche Erweiterungen, um Processing mit zusätzlichen Funktionen zu versehen, welche für den täglichen Gebrauch außerhalb von Visualisierung nötig sind. Im Laufe der Zeit wurde Processing performanter und von OpenGL, GLSL Shaders und GStreamer unterstützt.

Die wichtigsten Fakten zu Processing sind [54]:

- Open source
- Interaktive Visualisierungen in 2D, 3D mit Ausgabe in PDF und SVG
- OpenGL Integration
- Für Linux, Mac OSX, Windows, Android und ARM

- Zahlreiche Erweiterungen

Die Einsatzgebiete von Processing sind in der Kultur und der Wissenschaft zu finden [54]. Designer, Künstler und Architekten benutzen Processing in ihrer Arbeit. Projekte im Museum of Modern Arts in New York, Victoria und Albert Museum in London und viele weitere benutzen Processing. Es wird auch in Musik- und Tanzaufführungen benutzt und in Magazinen eingesetzt. In der Wissenschaft wird Processing für die Schaffung von Prototypen und Datenvisualisierungen eingesetzt. Für Visualisierungen wird es auch in der Industrie benutzt (Google, Intel, Yahoo, New York Times…).

Das Wichtigste für das Processing Projekt ist allerdings:
"But the most important thing about Processing and culture is not high-profile results – it's how the software has engaged a new generation of visual artists to consider programming as an essential part of their creative practice" [54].

2.8.3 P5 JS

P5.js ist eine Javascript Bibliothek mit dem selben Ziel wie Processing, die Schaffung einer einfachen Möglichkeit zur Visualisierung für Programmierneulinge. Aus diesem Grund passt diese Bibliothek gut in die vorgestellte Unterrichtssequenz. Die Idee hinter P5.js ist die Skizzierung von Visualisierungen und schnelles Prototyping, was perfekt zur Optimierung der Wissensvermittlung mit grafischer Unterstützung geeignet ist, da sie leicht verständlich ist. Das Arbeiten mit Code ist speziell für Künstler, Designer, Lehrende und Anfänger einfach. Mit P5.js kann jegliche Art von Grafik erzeugt werden. Es benutzt die Javascript CANVAS Funktionalität zum Zeichnen und die ganze Homepage wird zu einem Sketch. Mit Erweiterungen der P5.js sind Interaktion, Eingabe, Webcam, Sound und vieles mehr verfügbar.

Processing.js ist der Vorläufer von P5.js beziehungsweise eine Abspaltung von P5.js. Die Intention hinter dem Projekt processing.js ist, Lernenden beim Programmieren lernen zu helfen, ähnlich wie bei P5.js. Die offizielle Unterstützung von der Processing Foundation hat jedoch P5.js. Deshalb wird P5.js bei der grafisch basierten Programmierung als visuelle Repräsentation eingesetzt.

2.8.4 Entwicklerwerkzeuge

Für die Gestaltung eines effektiven Unterrichts wird versucht, jeden Störfaktor zu minimieren beziehungsweise komplett auszuschalten. Die Wahl der geeigneten Programmiertools könnte hier ein solcher Störfaktor sein, speziell beim Codeeditor. Der Codeeditor, welcher nur zum Schreiben des Programms genutzt wird, sollte für Einsteiger nicht zu

komplex sein. Eine IDE (Integrated Development Environment), welche Unterstützung wie Debugging oder Auto generated Code zur Verfügung stellt, verwirrt einen Programmierneuling [42]. Deshalb wurde bei der Wahl des Editors auf einen Texteditor zurückgegriffen, der flexibel im Einsatz ist, die wichtigsten Funktionen für ein einfaches Programmieren unterstützt und die Übersichtlichkeit für den Lernenden gibt. Als Texteditor wurde Notepad++ verwendet.

Abbildung 4: Notpad++

2.8.4.1 Editor

Der Editor „Notepad++" besitzt alle essentiellen Eigenschaften eines modernen Editors. Der Editor ist unter der GPL License und ist somit frei für Windows verfügbar. Er ist in C++ geschrieben und benutzt die Win32_API zur Steuerung. Eine kurze Übersicht wird in untenstehender Grafik gegeben. Die Oberfläche ist so aufgebaut, dass eine Bearbeitungsleiste zur Verfügung steht, ein Bereich für den Code und eine Ansicht der Zeilennummern.

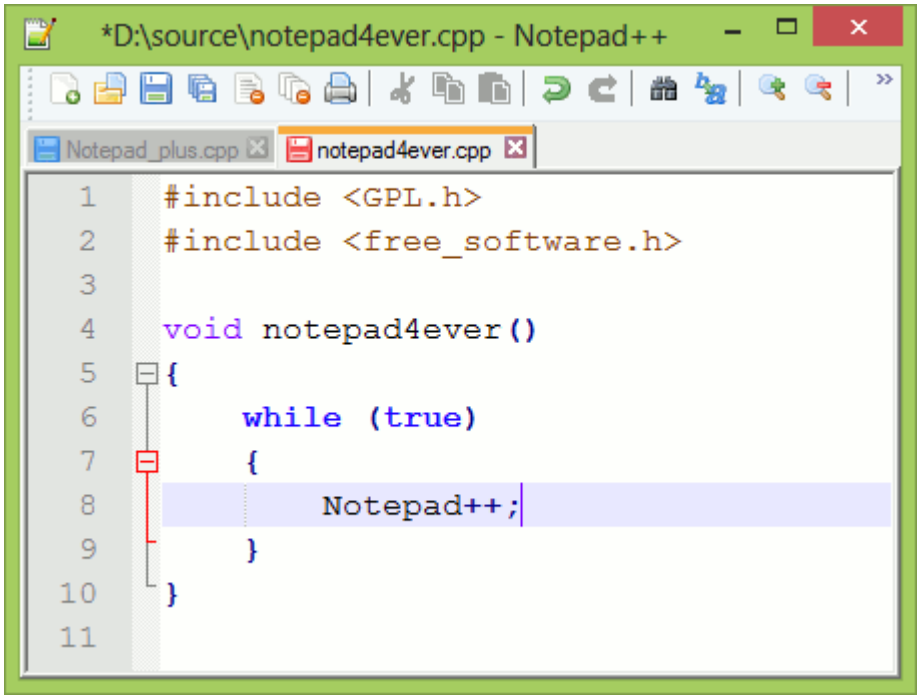

Abbildung 5: Notepad - Aufbau

Es kann mit dem Editor folgendes gemacht werden:

- Tastsenkürzel können benutzt werden,
- es gibt Syntax Highlighting und Syntax Folding
- Autocompletion ist möglich,
- Öffnen von mehreren Dokumenten gleichzeitig ist möglich,
- Zoom in und Zoom out Funktion ist vorhanden,
- die Dateiverwaltung kann einfach im Editor vorgenommen werden,
- Suchen und ersetzen kann durchgeführt werden,
- ein Vorwärts- und Rückgängigmachen des geschriebenen Codes ist auch vorhanden.

Diese Funktionen sind für das Lernen von Programmieren sinnvoll: Tastenkürzel dienen zum schnellen Arbeiten wie schnelles Abspeichern, ein neues File aufmachen und Ähnliches. Syntax Highlighting dient zum schnellen Erkennen von Strukturen mit Hilfe von Signalwörtern. Autocompletion vervollständigt schnell längere Wörter, sodass die Geschwindigkeit beim Programmieren erhöht werden kann. Die Zoom in und Zoom out Funktion hilft dem Lernenden, die optimale Größe für den geschriebenen Code zu erlangen. Die Dateiverwaltung dient zum raschen Öffnen und Schließen von Dateien. Mehrere Dokumente können gleichzeitig geöffnet sein, was die Übersichtlichkeit über die benutzten Dateien erhöht. Das Suchen und Ersetzen hilft beim Verändern von Variablennamen. Die Steuerung des Editors anhand einer internen „History", vorwärts und rückwärts, hilft beim Testen von Programmveränderungen.

2.8.4.2 Browser

Die Wahl des Browsers ist für eine angenehme Programmierung von Javascript nötig. Der Browser sollte die Möglichkeit besitzen:

- Javascript auszuführen,
- Entwicklerwerkzeuge aufzumachen, um den ausgeführten Code in der Javascript Console zu kontrollieren,
- Den Seitenquellentext zu untersuchen, um HTML Elemente zu betrachten

Ein möglicher Browser, der diese Eigenschaften besitzt, ist der Browser von Google, Google Chrome, wie unten gezeigt.

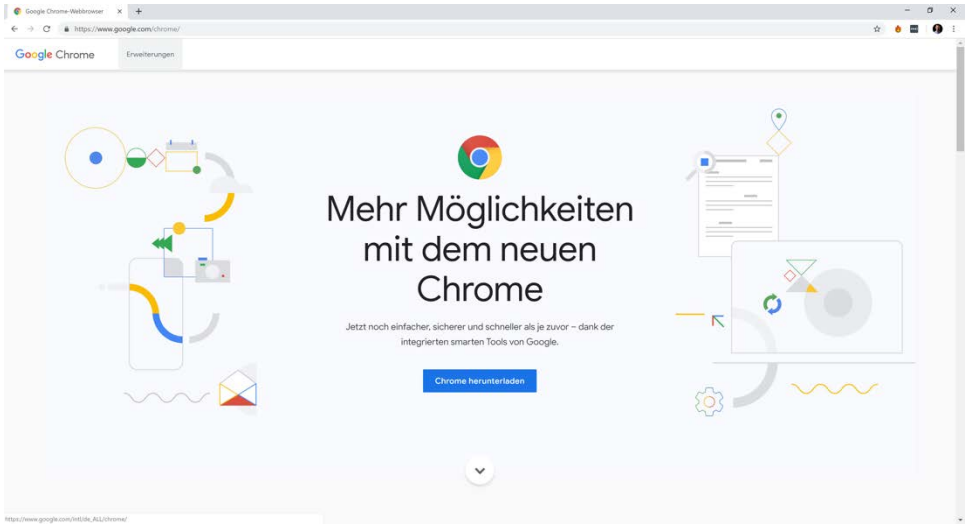

Abbildung 6: Chrome

Die Funktion des Browsers zur Ausführung von Javascript ist nötig, um überhaupt Javascript ausführen zu können. Je nach Bedürfnis müssen ECMAScript Varianten simuliert oder im Browser aktiviert werden. ECMAScript ist ein standardisiertes Javascript, welches von der ECMA International standardisiert wird. [57] Die Benutzung der „Developer Tools" Entwicklerwerkzeuge dient zur Ansicht des ausgeführten Codes in der Console für den textuellen Output. Das Browserfenster selbst mit der angezeigten Webseite ist wichtig für den visuellen Output, um gleiche Voraussetzungen bei den Entwicklerwerkzeugen zu haben und doch beim Output unterscheiden zu können. Die Ansicht des Seitenquellentexts ist für die Übersicht über die HTML Struktur der Webseite nötig.

2.9 Unterricht

Die Gestaltung eines Unterrichts ist vom Lehrer sorgfältig und mit Rücksicht auf die zu vermittelnden Lehrinhalte vorzunehmen. Deshalb wird in diesem Abschnitt auf den sinnvollen Aufbau von Unterricht eingegangen. Die Abhaltung eines korrekt geplanten Unterrichts ist für Lehrer und SchülerInnen zwecks Orientierung und Verständnis von Bedeutung. Der hier vorgestellte Aufbau des Unterrichts dient als Basis für die im nächsten Kapitel vorgestellten Unterrichtseinheiten.

Klassisches Programmieren lernen mit Frontalunterricht und Übungen auf textueller Basis beschäftigt sich mit dem Erlernen von Syntax und Algorithmen mit vorwiegend textuellem Output. Angefangen von einfachen Zuweisungen bis hin zu Schleifen, Arrays und Objects werden mit Hilfe eines Editors geschrieben und das Verstehen stellt sich dabei als sehr abstrakt für die SchülerInnen in der Oberstufe heraus.

Diese Arbeit setzt aus diesem Grund auf eine Verbesserung des „klassischen“ Unterrichts und vergleicht zwischen der klassischen Methode und einer grafisch basierten Methode. Die SchülerInnen bekommen in begrenzter Zeit die Grundkenntnisse und Grundkonzepte im Programmieren vermittelt, die Lernkurve wird hochgehalten und das Verständnis wird gefördert. Der grafisch basierte Unterricht hat die Vermittlung der Lehrinhalte mit bestmöglichem Output für Lehrer und SchülerInnen zwecks Erwerbung von Kompetenzen zum Ziel.

Die vermittelten Kenntnisse im Unterricht sind hier grob skizziert:

- Erlernen von Javascript
- Programmieren mit Javascript
- Lösungen für bestehende Programmierherausforderungen finden
- Suchen von Fehlern
- Erstellung eigenständiger Programme
- Verständnis für prozedurales, objektorientiertes und funktionales Programmieren

2.9.1 Aufbau der Lehrinhalte

Der Aufbau der Lehrinhalte hat zum Zweck, die abstrakt definierten Anforderungen des Lehrplans an den Unterricht umzusetzen und in praktische Fertigkeiten umzuwandeln und die Denkweise eines Programmierers zu erlernen. Die zwei erwähnten Unterrichtssequenzen sind strukturell gleich aufgebaut. Die Lehreinheiten laufen parallel zu einander und vermitteln in jeder Einheit dieselben Inhalte mit derselben Programmiersprache, aber unterschiedlich gewählten Ausgabe, was zum besseren Verständnis führen soll.
Der Aufbau der Lehreinheiten ist für alle Einheiten gleich gewählt. Dies wurde mit Rücksicht auf die Lernenden gewählt, um so wenig wie möglich Ablenkung vom Wesentlichen zu gewährleisten. Einheitlichkeit und Konformität im Ablauf sind für den Unterricht von hoher Bedeutung, da sich die SchülerInnen auf eine Struktur verlassen können.

2.9.2 Einzelstunden und Unterrichtssequenz

Die Grundlage für eine Unterrichtssequenz ist der gültige Lehrplan [44]. Zu Beginn wird der zu lehrende Stoff analysiert und ein Stoffverteilungsplan wird erstellt. Anschließend werden die einzelnen Unterrichtsstunden basierend auf der Stoffverteilung entwickelt.

2.9.3 Unterrichtsplanung

Das Thema der Unterrichtsstunde bestimmt den Aufbau und den Ablauf der einzelnen Stundenteile. Der didaktische Aufbau der Stunde wird passend zum Thema ermittelt und

die Kompetenzen, welche im Unterricht gelernt werden sollen, werden bestimmt [44]. Sinnvoll sind drei bis fünf Lernziele pro Stunde. Der zeitliche Aufbau mit ausreichend Zeitpuffer für Übungen sollten mit den Inhalten abgestimmt werden [44].

Die Aufteilung einer Unterrichtsstunde sollte wie ein Kapitel eines Buches sein, mit Abwechslung im Tempo, Ruhe- und Haltepunkten zur Orientierung und Schaffung von Aufmerksamkeit [45]. Der genaue Ablauf und die gewählte Methodik werden durch den Lehrer bestimmt. Die notwendige Transparenz hilft den SchülerInnen sich zu orientieren, denn SchülerInnen lernen besser, wenn sie wissen, worum es geht. Die Themen sollten schülergerecht formuliert sein, sodass sich die SchülerInnen etwas vorstellen können, jedoch mit deutlichem Bezug auf den Inhalt gewählt werden [45].

2.9.4 Skizze sinnvoller Unterrichtsphasen

Dieser Punkt ist für das eindeutige Verständnis einer passenden Unterrichtsstunde von Bedeutung. Weiter soll sichtbar werden, welche Tätigkeiten beachtet wurden, um die vorgestellte Methode des grafisch basierten Programmierens durchzuführen. Es wird ein Beispiel für sinnvolle Unterrichtsphasen gegeben [45]:

1. Warm Up
2. Das Stundenprogramm
3. Ermittlung des Vorwissens der SchülerInnen zum Thema
4. Info- und Unterrichtsgespräch
5. Arbeitsauftrag
6. Selbstständige Arbeit
7. Ergebnisse präsentieren
8. Stundenschluss

Das Warm Up beschäftigt sich mit dem Heranführen der SchülerInnen an die Unterrichtsstunde. Das Thema der Stunde wird verkündet. Mit entsprechender Bezeichnung des Themas kann Interesse bei den SchülerInnen erzeugt werden.

Als nächstes wird das Wissen der SchülerInnen zum Thema erfragt [45]. Durch die Wiederholung der letzten, gehaltenen Stunde wird das Wissen der SchülerInnen aufgefrischt und Erfolge fördern die Motivation. Bei neuen Problemstellungen und Inhalten wird am Beginn der Stunde das Wissen zu einem bestimmten Thema abgefragt. Je nach Vorerfahrung könnte die Befragung der Klasse in einer Gruppenarbeit münden und somit könnte der Unterricht aktiv gestaltet werden – präsentieren, aufschreiben und zusammentragen. Anschließend werden die Ausarbeitungen analysiert und offene Punkte geklärt sowie die Kompetenzen ermittelt. Ein weiterer Einstieg in den Unterricht kann auch mit Hypothesen

erfolgen, welche durch die SchülerInnen geprüft werden können [45]. Die Befragung der SchülerInnen zu Beginn kann schnell uninteressant wirken, wenn nur wenige SchülerInnen in Kontakt mit dem Lehrenden treten und teilnehmen. Hier ist es von Bedeutung, Informationen zu liefern und die SchülerInnen durch ein Gespräch zu binden.

Arbeitsaufträge sollen auf Anhieb verständlich und nachvollziehbar sein. Unter der Zuhilfenahme von visuellen Hilfsmitteln wird ein Arbeitsauftrag unterstützt, die Ergebnisse werden präsentiert und erklärt.

Zu jeder Stunde gehört eine Phase der selbstständigen Arbeit. Informationen zu Themen können recherchiert werden, allein oder in Gruppen. Selbstständiges, aktives und erfolgreiches Lernen ist mit Austausch zwischen den SchülerInnen möglich und fördert das Lernen. Die Ergebniskontrolle soll für alle SchülerInnen ermöglicht werden. Eine Kontrolle im Plenum ist schwierig. Zu erfolgreichem, selbstständigem Arbeiten zählen Literaturauswertung, Rechercheplan, Interviews vorbereiten, Rollenspiele üben oder Präsentation vorbereiten [45].

Die Präsentation der Ergebnisse ist eine herausfordernde Tätigkeit. Die SchülerInnen müssen sich alle gleichzeitig konzentrieren. Weiters gehört die Analyse des Ergebnisses ordentlich geplant. Die Motivation für Präsentationen von Ergebnissen sinkt in höheren Klassen.

Der Abschluss einer Stunde ist der wichtigste Bereich der Schlussphase. Die Stunde sollte nicht den Eindruck vermitteln, dass sie zerfließt, denn dieser Eindruck vermittelt Unwichtigkeit. Ein Teil des Schlussrituals sind die Hausaufgaben. Von höherer Bedeutung sind allerdings Feedback und eine kurze Wiederholung über das Gelernte in der Stunde.

3 Ablauf des Vergleichs der Unterrichtssequenzen

Der Vergleich der Unterrichtssequenzen basiert auf den zuvor recherchierten wissenschaftlichen Artikeln. Diese wissenschaftlich fundierte Vorgehensweise ist für eine signifikante Aussage wichtig.

3.1 Einteilung der Gruppen für den Unterricht

Für die Testung der Unterrichtsmethode werden zwei Gruppen benötigt, eine Testgruppe mit grafischer Programmierung und eine Kontrollgruppe mit klassischer Programmierung. Für den Vergleich wurden insgesamt 32 Personen im Alter zwischen 15 und 16 Jahren aus dem Oberstufenrealgymnasium und dem Wahlpflichtfach Informatik herangezogen. Die Einteilung der Personen in die 2 Gruppen erfolgte quasi zufällig, und zwar alphabetisch nach den Nachnamen. Frauen und Männer wurden auf die Gruppen gleich aufgeteilt, sodass in jeder Gruppe 10 Männer und 6 Frauen sind. Die Einteilung berücksichtigt also keine Vorkenntnisse und Fertigkeiten.

3.2 Einstiegstest

Der Einstiegstest dient zur Ermittlung der Fähigkeiten und Fertigkeiten der SchülerInnen im Bereich der Programmierung und hat keinen Einfluss auf die Einteilung der Gruppen. Er zielt auf Vorwissen in der IT ab und prüft die Fähigkeiten in logischem und mathematischem Denken ab. Dieser Test wurde durch aktuelle Einstufungstests für Fachinformatiker in Deutschland und das Studium der Informatik beeinflusst [44, 45]. Durch die Vorstufe, die 5. Oberstufe im Gymnasium, können gewisse Kenntnisse am Computer vorausgesetzt werden. Ein ernstzunehmendes Problem für einen effektiven Unterricht im Bereich der Programmierung ist die Schreibgeschwindigkeit auf der Tastatur. Ein Zehnfingersystem beim Schreiben auf der Tastatur ist Pflicht für schnelles Vorankommen im Informatikunterricht. Sind SchülerInnen nicht geübt, kann ein Unterricht nur sehr zäh und aufgrund fehlender Geschwindigkeit beim Tippen mit hoher Verzögerung durchgeführt werden. Darunter leidet bei den meisten schnellen Tippern die Motivation im Unterricht und die Aufmerksamkeit und die damit verbundene Aufnahmefähigkeit sinkt drastisch. Aus diesem Grund wurde auf ähnliche Geschwindigkeit bei den SchülerInnen bei der Durchführung des Unterrichts geachtet.

Der Einstiegstest zur Ermittlung der Kenntnisse in den Bereichen IT, Logik und Mathematik ist als Multiple Choice konzipiert und kann am Ende der Arbeit gefunden werden. Die Ergebnisse des Tests sollen hier kurz beschrieben werden und die Ausgangssituation der einzelnen SchülerInnen gezeigt werden [24]. Die Erkenntnisse aus dem Einstiegstest sind wichtig, um anschließend eine Aussage über die Effektivität der Ausbildungsmethode treffen zu können.

3.2.1 EDV

Die Ergebnisse der EDV-Fragen zeigen folgendes Bild:

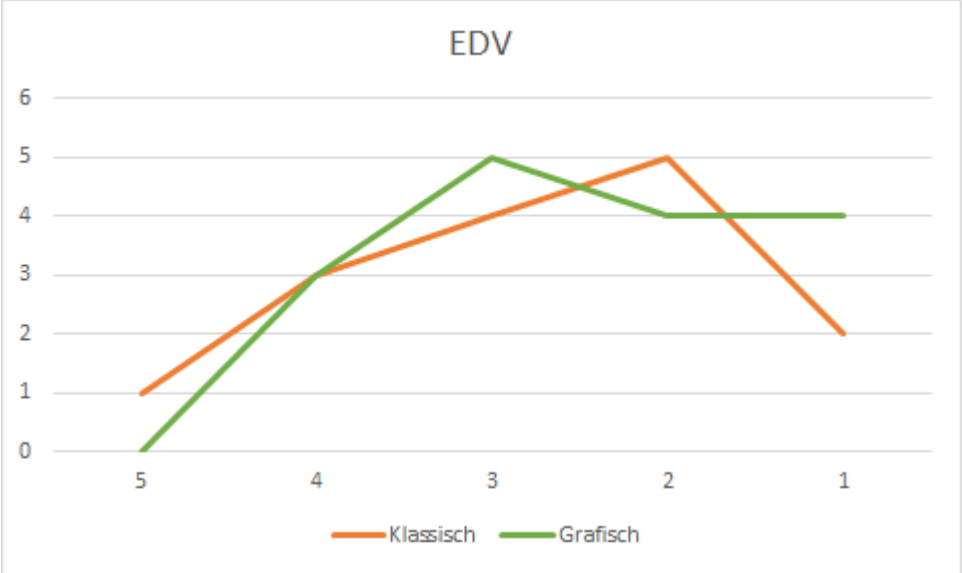

Abbildung 7: Noten der EDV-Testung

Beide Gruppen haben eine ähnliche Verteilung der Noten für den Einstiegstest. Die „grafische" Gruppe hat allerdings eine in Richtung der Note Gut tendierte Verteilung, was bedeutet, dass etwas mehr Wissen in der EDV vorhanden ist.

3.2.2 Mathematik

Die Verteilung in der Mathematik hat eine in Richtung Genügend tendierende Verteilung. Dies bedeutet, dass die Mathematikaufgaben schlecht gelöst wurden.

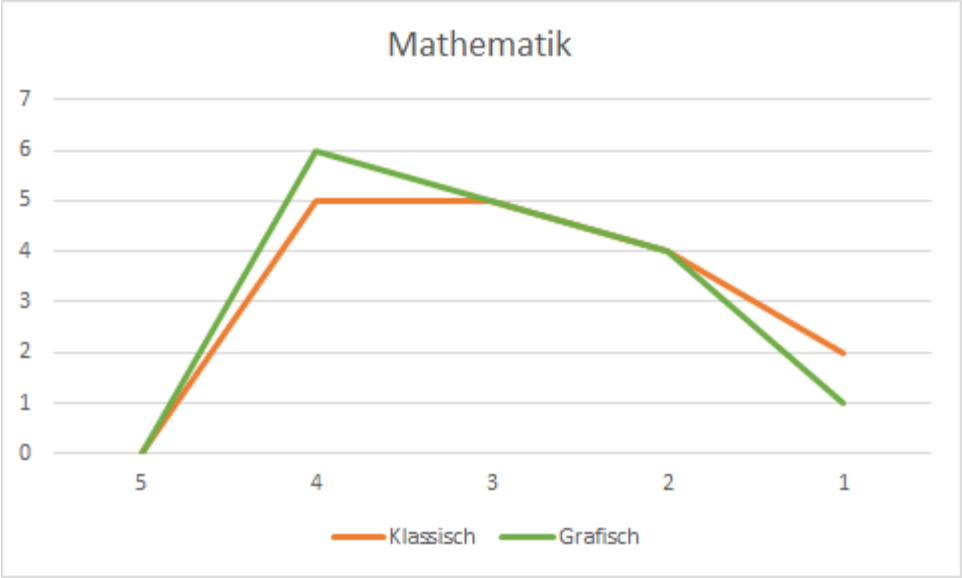

Abbildung 8: Noten der Mathematik-Testung

Viele SchülerInnen hatten Probleme mit den Hochzahlen, Binärzahlen und Hexadezimalzahlen. Aus diesem Grund ist das Ergebnis im Bereich Mathematik bei beiden Gruppen schlecht gewesen. Die Verteilung der Noten ist nahezu gleich, was an ähnlichen Kurven abgelesen werden kann.

3.2.3 Logik

Im Bereich Logik differieren die Kenntnisse der beiden zu testenden Gruppen. Bei der „klassischen" Gruppe sind vorwiegend Ergebnisse um die Note Befriedigend und bei der „grafischen" Gruppe sind die Mehrheit Gut und Sehr Gut.

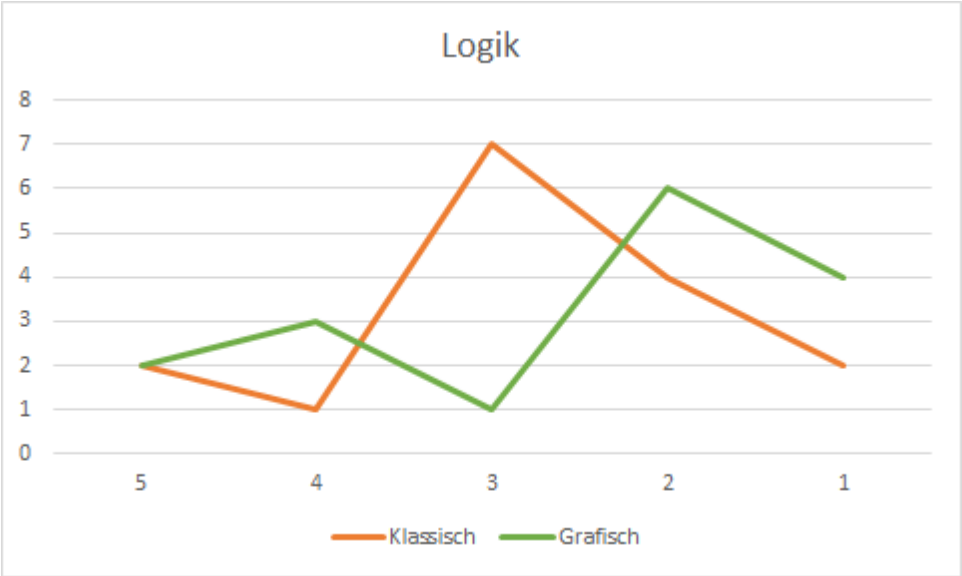

Abbildung 9: Noten der Logik-Testung

Durch das bessere logische Verständnis kann ein möglicher Vorteil im Verständnis für das Programmieren abgeleitet werden, allerdings ist zu beachten, dass die Mittelwerte der Ergebnisse in der „grafischen" und „klassischen" Gruppe nahezu gleich sind. Dies ist aus der Verteilung der Ergebnisse in obiger Grafik deutlich zu erkennen. Die Mittelwerte aller Ergebnisse aus den drei Bereichen des Vortests der beiden Testgruppen differieren um 0,06, liegen also sehr nahe beeinander. Somit sind nahezu gleiche Voraussetzungen in beiden Gruppen gegeben.

3.3 Aufbau der Unterrichtssequenzen

Der Aufbau der Unterrichtssequenzen für die Bereiche „klassisches" Programmieren und „grafisches" Programmieren werden in diesem Abschnitt skizziert. Beide Unterrichtssequenzen beinhalten denselben Lehrinhalt, deren Anordnung ist allerdings etwas unterschiedlich. Dies rührt daher, dass die Bibliothek p5.js „anders" eingeführt werden muss als die textuell basierte Programmierung des traditionellen Unterrichts. Die Anwendung von Funktionen, ohne zu wissen, was genau Funktionen sind, muss bei p5.js vorgezogen werden.

3.3.1 Unterrichtssequenzen in traditioneller Art

Bei der Unterrichtssequenz in traditioneller Art wird folgende Lehrinhaltsverteilung vorgenommen:

1. Vorbereitungseinheit, Einführung in das Programmieren (Eingabe-Ausgabe, Anwenden von Funktionen,…)
2. Variablen und Zuweisungen, Primitive Datentypen (Konvertieren), String-Operatoren, Number-Operatoren
3. Kontrollstrukturen: Operatoren und Vergleiche, Schleifen
4. Funktionen und Scope von Variablen
5. Komplexe Datentypen (Objekte), Private Eigenschaften, Vererbung
6. Komplexe Datentypen (Arrays),
7. Algorithmen, Programmierung einer Sortierung, Funktionale Programmierung
8. Lösung einer Aufgabe

3.3.2 Unterrichtssequenzen mit visueller Unterstützung und p5.js

Bei der Unterrichtsgestaltung mit grafisch basierter Ausgabe werden dieselben Inhalte wie oben bereits beschrieben vermittelt, mit dem Unterschied, dass bei Punkt 2 zusätzlich die Funktionen zur Steuerung von p5.js erklärt werden.

3.3.3 Aufbau der Unterrichtseinheiten in der vorgestellten Methodik

Zum besseren Verständnis der im nächsten Kapitel vorgestellten Unterrichtseinheiten wird an dieser Stelle der Aufbau einer Unterrichtseinheit beschrieben. Der Aufbau der Unterrichtseinheiten der vorgestellten Unterrichtsmethodik ist für das „klassische" Programmieren und das „visuelle" Programmieren gleich. Beide Unterrichtssequenzen haben dieselben Lehrinhalte, wie oben gezeigt, der Ablauf einer einzelnen Einheit ist wie folgt aufgeteilt:

1. Einleitung
2. Wiederholung der letzten Stunde
3. Theorie
4. Beispiele und Übungen
5. Kontrollfragen
6. Zusammenfassung
7. Weiterführende Übungen

Die Einleitung und Wiederholung der letzten Stunde wird in den Unterrichtseinheiten nicht explizit beschrieben. Hier gilt es, eine geeignete Form für den Einstieg in die Unterrichtsstunde, wie zuvor kurz skizziert, zu finden. Dies ist nicht Fokus dieser Arbeit. Genauso wird auf die Zusammenfassung während des Unterrichts nicht eingegangen. Genauer erklärt werden hingegen die vermittelte Theorie, Beispiele und Übungen sowie Kontrollfragen und weiterführende Übungen, da diese Punkte den Kern der Arbeit darstellen.

Die grundlegende Konzeption hinter den folgenden Unterrichtseinheiten ist: Erklärung des Lehrinhalts der Stunde mit passender textueller oder grafischer Präsentation von Erklärungen und Durchführung von Übungsbeispielen thematisch orientiert am Lehrinhalt der Stunde. Die Erstellung einer Lehreinheit basiert auf der Analyse des zu vermittelnden Inhalts, der Prüfung und Überlegung wie die Inhalte grafisch oder textuell ausgegeben werden, der Erstellung von Übungen zum Lernen und zur Kontrolle der zu vermittelnden Inhalte für die SchülerInnen. Dieser Prozess zur Erstellung von Lehreinheiten ist für alle unten stehenden Einheiten durchgeführt worden.

3.4 Vorbereitungseinheit

In der Vorbereitungseinheit werden die benötigten Tools erklärt, welche für die darauffolgenden Einheiten benötigt werden. Javascript als Programmiersprache wird vorgestellt und die Entwicklerwerkzeuge werden geöffnet.

Abbildung 10: HTML5

Zu Beginn des Unterrichts werden die SchülerInnen an die neue Thematik Programmieren herangeführt. Sie werden im Plenum nach Programmiererfahrungen befragt. Danach wird eine Übersicht über den Editor Notepad++ gegeben und Chrome als Tool für die Ausgabe mit der Console und der angezeigten Webseite vorgeführt. Den SchülerInnen wird ein Beispielprogramm demonstriert. Jede(r) SchülerIn bekommt eine HTML Datei. Der Aufbau der HTML Datei wird kurz erklärt und kann wie folgt dargestellt werden:

```
1  <html>
2      <head>
3          <title>
4              LEARNING PROGRAMMING
5          </title>
6      </head>
7      <body>
8          <script>
9              console.log("HELLO WOLRLD!");
10         </script>
11     </body>
12 </html>
```

Abbildung 11: Einführung in HTML

Die Einbindung von Javascript kann an unterschiedlichen Stellen im HTML-Code stattfinden. Danach wird die Ausführung von Code in den DevTools gezeigt. Das Aufmachen des Chrome Browsers wird gezeigt und der Refresh der Webseite zum neuerlichen Ausführen des Codes bei Abänderung wird demonstriert. Mit der alert() und console.log() Funktion von Javascript werden bereitgestellte Funktionen erklärt und der Aufruf der Funktionen wird beschrieben. Durch dies sollen die Studierenden die Fähigkeit erlernen, integrierte Funktionen zu erkennen und diese ausführen zu können. Als nächstes werden den SchülerInnen die Begriffe Syntax und Semantik (reserved words) erklärt bzw. es wird der Begriff nochmals wiederholt. Das Arbeiten mit dem Editor wird gezeigt, ebenso wie das Speichern und mehrfache Öffnen der Dateien. Das Syntax Highlighting wird vorgeführt und den

SchülerInnen nähergebracht. Es werden die benötigten HTML Tags erklärt und der Unterschied zwischen Programmierung und Strukturierung von Inhalten (HTML/XML) wird beigebracht. Weiters werden die Zeilennummern in Notepad++ erklärt. Kurz wird das Integrated Development Environment beschrieben und andere Programmiersprachen, das Kompilieren und Interpretieren von Code sowie Maschinencode werden in diesem Zusammenhang erläutert.

Als Übung legen die SchülerInnen ein neues HTML File an und schreiben eine Grundstruktur mit HTML und führen im Script Tag ein console.log aus. Bei dieser Übung ist gegenseitiges Helfen erlaubt.

Am Ende werden Kontrollfragen zur Stunde gemacht und die Stunde wird mit einer Zusammenfassung geschlossen.

3.5 Lehreinheiten

Bei den klassischen Lehreinheiten wird das Programmieren mit Hilfe des textuellen Outputs gelehrt. Dieser ist im Gegensatz zum visuellen Output etwas schwieriger zu erlernen. Der textuelle Output des Programmierens erfolgt in der Browser-Console in Chrome. Der zentrale Fokus bei den Lehreinheiten liegt auf der Ausgabe. Das Begreifen fällt dem Lernenden leichter, wenn eine visuelle Repräsentation des Lehrinhalts vorhanden ist. Wie bereits oben beschreiben, merken wir Menschen uns besser Illustrationen als Geschriebenes. Mit diesem Wissen werden die Einheiten aufgebaut und die Ausgabe dargestellt. Die Einheiten werden beschrieben, die Vorgehensweise wird dadurch erklärt, die vermittelten theoretischen Begriffe und Konzepte werden hervorgehoben und die Beispiele mit den klaren Differenzen in der Ausgabe werden skizziert.

3.5.1 Einheit 1

Der Einstieg in der Einheit 1 basiert auf den Grundlagen, welche in der Vorbereitungseinheit geklärt wurden, welche für alle SchülerInnen gleich waren.

3.5.1.1 Theorie

In der ersten Theorieeinheit werden folgende Begriffe erklärt. Diese Begriffe sind die zentralen Elemente des ersten Bereichs.

1. Variable - CamelCase, Case-sensitive
2. Zuweisen, Initialisieren und Deklarieren
3. Kommando
4. Datentypen

5. Operatoren
6. Konvertieren
7. Vergleiche

3.5.1.2 Beschreibung

Die ersten eigenen Schritte im Programmieren werden in dieser Einheit unternommen. Die signifikanten Stichwörter in dieser Lektion sind: Variablen und Zuweisungen. Variablen zu verstehen und die Benutzung dieser mit Hilfe von Zuweisungen ist das essentielle Element in der Programmierung. Bei den Variablen wird der Aufbau einer Zuweisung mit dem Abschluss des Kommandos mit einem Strichpunkt erklärt. Ziel ist es, den SchülerInnen unterschiedliche Definitionen beizubringen, sodass das Lesen von fremdem Code erleichtert wird. Dies ist für die Vorführung der Beispiele im Unterricht nötig.
Bei der Zuweisung von Werten werden unterschiedliche Datentypen eingesetzt. Die SchülerInnen lernen dadurch, dass es unterschiedliche Möglichkeiten gibt, wie Werte vorliegen können und deutlich differenziert werden muss, auf welche Art und Weise ein Wert existiert. Davon abhängig sind klarerweise die Operationen und Funktionen. Das ist einer der Schlüsselpunkte, der von den Lernenden verstanden werden muss. Mit den Datentypen eng in Kontakt steht der typeof Operator, welcher die Datentypen ausgeben kann. Für SchülerInnen ist es oft schwer begreifbar, dass Variablen überschrieben werden können und auch passend dazu in Javascript ihren Datentyp ändern. Die dynamische Typisierung von Variablen zur Laufzeit müssen den SchülerInnen bei der Programmierung erklärt werden. Auch ein essentieller Punkt für SchülerInnen ist: die Variable muss ein einziges Mal angelegt werden und kann im Code öfters mit Werten überschrieben werden, ohne ein weiteres Mal das Keyword „var“ benutzen zu müssen.
Die ersten Beispiele werden an dieser Stelle vorgeführt. Anschließend werden die ersten Übungen zum Programmieren gestartet. Nach dieser Übung sollten die Datentypen boolean, string, null, undefined und number bekannt sein. Sowohl die Begriffe Camelcase, Initialisieren, Deklarieren, Escape-Charakter, Case-sensitive als auch der Aufbau eines Variablennamens kommen in dieser Einheit vor.
Als Nächstes werden Kommentare erklärt und ihre Bedeutung für Lesbarkeit und Dokumentation von Code sowie das Ausblenden von Codeausführung und Debugging wird beschrieben. Nach der Dokumentation werden die Vergleichsoperatoren und Kontrollstrukturen in Javascript eingeführt. Die Einführung der Vergleichsoperatoren ist einfach, da den SchülerInnen die unterschiedlichen Operatoren aus dem Mathematikunterricht bekannt vorkommen. Eine Herausforderung bei manchen SchülerInnen war es, zu erklären, dass nur Wahrheitswerte Ergebnisse aus Vergleichen sind. Die Operatoren “===” und “!==” verlangt eine präzise Erklärung der Bedeutung. Mit den Operatoren wird auch die Priori-

tät der Operatoren für die Programmierung beschrieben. Diese ist in folgender Grafik ersichtlich und die vermittelnden Operatoren werden tabellarisch in der Grafik gezeigt.

Pre	Operator	Pre	Operator
1	() [] . x++ x--	8	&
2	! ~ ++x --x	9	^
3	* / %	10	|
4	+ -	11	&&
5	<< >> >>>	12	||
6	< <= > >=	13	= op=
7	== != === !===	14	,

Abbildung 12: Operatoren in Javascript [56]

Anschließend werden Übungen von den SchülerInnen durchgemacht, um das Gelernte gleich selbst umzusetzen. Das Konvertieren zwischen den unterschiedlichen Datentypen ist in dieser Einheit auch als Übung vorgesehen, zum Beispiel wird ein Zahlenwert als String abgespeichert und die Konvertierung wird vorgezeigt und geübt. Das letzte Beispiel mit dem typeof Operator (Abbildung 21) zeigt dem Schüler, der Schülerin, die Datentypen und abschließende Übungen zur Konvertierung wiederholen die zuvor gelernte Theorie in der Praxis.

Für die grafische Ausgabe des Programmcodes ist für die SchülerInnen der „grafischen“ Gruppe folgendes zu erklären: der Browser ist eine Zeichenfläche mit dem Signalwort “canvas”, welcher seit HTML5 eingesetzt wird. Die SchülerInnen erkennen die Ähnlichkeit der Zeichenfläche mit einem Koordinatensystem, bei dem sich der Ursprung in der linken oberen Ecke befindet.

Als nächstes gilt es den Lernenden die Funktionen setup und draw von der Bibliothek p5.js vorzustellen. Ein ähnliches Vorgehen wird im Buch „Programmieren lernen mit Computergrafik“ durchgeführt [58]. Gemeinsam wird ein Objekt gezeichnet und ein kurzer Hinweis auf das Verändern und Anpassen der Farbe durch strokeColor, strokeWeight, fill wird gegeben. Die SchülerInnen unternehmen erste Schritte mit p5.js.

3.5.1.3 Beispiele textuell

Die Beispiele der ersten Einheit mit textueller Ausgabe werden an dieser Stelle beschrieben. Neben dem Beispiel wird die Ausgabe bei den Beispielen gezeigt, um dem Leser bzw. der Leserin zu zeigen, worin der Unterschied im Erfassen der Lehrinhalte besteht. Das erste Beispiel wird wie folgt dargestellt:

```
<html>
    <head>
        <title>
            LEARNING PROGRAMMING
        </title>
    </head>
    <body>
        <script>
            var snowBoard = 'Burton';
            var hasSnowBoard = true;
            var numOfSnowBoards = 0;
            var snow;
            var mySnowBoard = null;
            console.log(snowBoard);
            console.log(typeof snowBoard);
            console.log(hasSnowBoard);
            console.log(typeof hasSnowBoard);
            console.log(snow);
            console.log(typeof snow);
            console.log(mySnowBoard);
            console.log(typeof mySnowBoard);
        </script>
    </body>
</html>
```

Abbildung 13: Beispiel1 (klassisch): Variablen

Ein geeigneter einfacher Einstieg ist für die Lernenden bezüglich der Motivation entscheidend, da die Aufnahmefähigkeit der SchülerInnen davon abhängt. Das erste Beispiel in Abbildung 13 zeigt mögliche Datentypen in Javascript und die Ausgabe erfolgt mittels console.log in der Console des Browsers. Es werden Zuweisungen und Deklarationen von Variablen vorgeführt. Die Ausgabe erfolgt textuell wie in Abbildung 14.

```
Burton                    index.html:14
string                    index.html:15
true                      index.html:16
boolean                   index.html:17
undefined                 index.html:18
undefined                 index.html:19
null                      index.html:20
object                    index.html:21
```

Abbildung 14: Ausgabe des Beispiel 1

In der Live-Demonstration werden in der Console direkt Änderungen vorgenommen. Dadurch wird gezeigt, wie dynamisches Typisieren in Javascript funktioniert und dass eine Variable ihren Typ rasch ändern kann. Zum Beispiel wird der Wert von hasSnowboard von einem boolean Wert auf einen number Wert gesetzt wird und mittels typeof Operator und der textuellen Ausgabe werden die Datentypen angezeigt. Beispiele für falsche Namensgebungen folgen und schaffen bei den Lernenden Bewußtsein.

```
23          var 3SnowBoards = 3;
24          var has Snowboard = false;
25          var my@Board = 'Burton';
```

Abbildung 15: Variablen: falsche Definition

Das zweite Beispiel zeigt den Umgang mit Operatoren und kann wie in Abbildung 15 visualisiert werden. Das Verständnis von Operatoren ist aufgrund der mathematischen Operationen für SchülerInnen einfach zu verstehen. Mit arithmetischen Operatoren wird begonnen, um die Ähnlichkeit zur Mathematik zu zeigen und die SchülerInnen werden durch Bekanntes motiviert. Dies fördert die Aufnahme von Neuem und die Verständlichkeit ist gegeben.

```
1  <html>
2      <head>
3          <title>
4              LEARNING PROGRAMMING
5          </title>
6      </head>
7      <body>
8          <script>
9              var num1 = 100;
10             var num2 = 10;
11             var result;
12             result = num1 + num2;
13             console.log(result);
14             result = num1 - num2;
15             console.log(result);
16             result = num1 / num2;
17             console.log(result);
18             result = num1 * num2;
19             console.log(result);
20             result = num1 % num2;
21             console.log(result);
22         </script>
23     </body>
24 </html>
```

Abbildung 16: Console und arithmetische Operatoren

Weiters werden kombinierte Zuweisungen, usw. gezeigt (+=, -=, …). Die Ausgabe des obigen Beispiels aus Abbildung 16 führt zu folgender Ausgabe:

Abbildung 17: Ausgabe der arithmetischen Operatoren

Das dritte Beispiel, dargestellt in Abbildung 18, bringt den SchülerInnen Vergleichsoperatoren näher. Nach den arithmetischen Operatoren sind diese den SchülerInnen aufgrund der Logik aus dem Mathematikunterricht bekannt. Mit Hilfe dieser Operatoren lernen die SchülerInnen Werte zu vergleichen und sehen, dass das Ergebnis ein Wahrheitswert ist.

Eine Schwierigkeit in diesem Bereich besteht in der Erfassung der Vergleichsoperatoren === und !==, welche nicht nur auf Gleichheit des Wertes überprüfen, sondern auch auf den gleichen Datentyp kontrollieren. Dieser Unterschied ist für die Lernenden nicht sofort begreifbar.

```
<html>
    <head>
        <title>
            LEARNING PROGRAMMING
        </title>
    </head>
    <body>
        <script>
            var result;
            result = 2 == 2;
            console.log(result);
            result = 2 === 2;
            console.log(result);
            result = 2 == '2';
            console.log(result);
            result = 2 === '2';
            console.log(result);
            result = 2 < 3;
            console.log(result);
            result = 2 > 3;
            console.log(result);
            // <=, >=, !=, !==
        </script>
    </body>
</html>
```

Abbildung 18: Vergleichsoperatoren in Javascript

Die Ausgabe wird hier für das bessere Verständnis, was die SchülerInnen im Unterricht sehen gezeigt.

Abbildung 19: Ausgabe Vergleichsoperatoren

Das vierte gezeigte Beispiel in Abbildung 20 in dieser Einheit beschäftigt sich mit dem Konvertieren von Werten. Es werden die Objekte string, boolean und number verwendet, um Konvertierungen vorzuzeigen. Die Konvertierungen sind für das Verständnis von Datentypen für die SchülerInnen von hoher Relevanz. Am Ende des Beispiels wird der + Operator in Verbindung mit Zeichenketten vorgeführt, was den SchülerInnen das Verständnis von Operatoren in unterschiedlichen Kontexten vermittelt.

```
1  <html>
2      <head>
3          <title>
4              LEARNING PROGRAMMING
5          </title>
6      </head>
7      <body>
8          <script>
9              var result;
10             result = String(1000);
11             console.log(typeof result);
12             result = Number("1000");
13             console.log(typeof result);
14             result = Boolean("YES");
15             console.log(typeof result);
16             result = Boolean("");
17             console.log(typeof result);
18         </script>
19     </body>
20 </html>
```

Abbildung 20: typeof-Operator

Als Ausgabe ist hier folgendes zu sehen:

```
string                                                    ue4.html:11
number                                                    ue4.html:13
boolean                                                   ue4.html:15
boolean                                                   ue4.html:17
Ergebnis: false                                           ue4.html:19
```

Abbildung 21: Ausgabe typeof-Operator

Derselbe Inhalt wird in der Gruppe mit „grafischen“ Programmieren vermittelt. Die Beispiele sind aufgrund der Verwendung der Bibliothek p5.js zum Darstellen von Grafiken verschieden. Das Ziel, einen Vergleich zwischen textuellen und visuellen Output durchzuführen, wird durch dieselben vermittelten Inhalte ermöglicht.

3.5.1.4 Beispiele visuell

Bei den Beispielen mit grafischer Ausgabe werden eine Grundstruktur des ausführbaren Codes für p5.js gezeigt und die Struktur des Programms wird den SchülerInnen erklärt. Durch eine klare, einfache Struktur entsteht ein Verständnis bei den Lernenden für den Ablauf. Die Funktionen als Konstrukt in der Programmiersprache werden an dieser Stelle nicht näher erklärt. Die Aufgabe der Funktionen setup und draw sowie die Begrenzungen der Funktion durch die geschwungenen Klammern (curly brackets) werden beschrieben. Ein einführendes Beispiel für das Verständnis zeigt den Ablauf von p5.js. Die Struktur des Codes sieht wie folgt aus:

```
<html>
    <head>
        <title>
            LEARNING PROGRAMMING
        </title>
        <meta name="viewport"
            width=device-width,
            initial-scale=1.0,
            maximum-scale=1.0,
            user-scalable=0>
        <style> body {padding: 0; margin: 0;} </style>
        <script src="p5/p5/p5.min.js"></script>
        <script src="p5/p5/addons/p5.dom.min.js"></script>
        <script src="p5/p5/addons/p5.sound.min.js"></script>
    </head>
    <body>
        <script>
            function setup() {
                console.log('STARTING');
            }
            function draw() {
                console.log('RUNNING');
            }
            console.log('HELLO WORLD');
        </script>
    </body>
</html>
```

Abbildung 22: Beispiel 1 (grafisch): Einführung in Javascript

Die SchülerInnen lernen dadurch, wie Werte auch über die Console ausgegeben werden können, wichtiger ist jedoch den Ablauf von p5.js zu verstehen. Bei der Ausführung des Codes wird klar, dass „RUNNING” immer wieder ausgegeben wird und ein Teil des Pro-

grammablaufs sich immer wieder wiederholt. Ein Hinweis auf aktuelle Spiele und die Funktionsweise von „Game Loops" und wie die Programmierung bei Computerspielen funktioniert, hilft dem Schüler, der Schülerin beim Verständnis und motiviert zugleich. Es folgt eine Live Demonstration und ein Viereck wird gezeichnet. Der Code zwischen den script-Tags wird getauscht und sieht danach wie folgt aus:

```
17      <script>
18          function setup() {
19            createCanvas(1000, 800);
20            background(220, 220, 220);
21          }
22
23          function draw() {
24            rect(50, 100, 200, 100);
25          }
26      </script>
```

Abbildung 23: Funktionen für grafische Ausgabe in p5.js

Die Ausgabe ist im Browser, wie in Abbildung 24 zu sehen. Diese grafische Ausgabe ist im Fokus dieser Arbeit, sie bringt den Unterschied im Verständnis beim Lernenden.

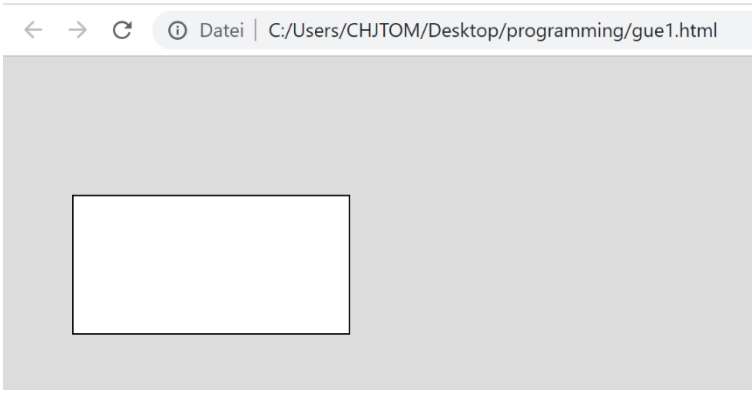

Abbildung 24: Ausgabe (grafisch): Einführung

Die Motivation bei den SchülerInnen steigt, wenn das gezeichnete Rechteck auf der Webseite im Browser angezeigt wird. Als nächstes ist die Einführung von Variablen in Abbildung 25 zu sehen.

```
<script>
    var widthOfCanvas = 1000;
    console.log(typeof widthOfRectAngle);
    var heightOfCanvas = 800;
    console.log(typeof heightOfRectAngle);

    //var 3SnowBoards = 3;
    //var has Snowboard = false;
    //var my@Board = 'Burton';
    var hasCanvas = true;
    console.log(typeof hasCanvas);
    var circle;
    console.log(typeof circle);
    circle = null;
    console.log(typeof circle);

    function setup() {
        var canvasName = 'Drawing area';
        console.log(typeof canvasName);
        createCanvas(widthOfCanvas, heightOfCanvas);
        background(220, 220, 220);
    }

    function draw() {
        rect(50, 100, 200, 100);
    }
</script>
```

Abbildung 25: typeof-Operator

Die Deklarationen der Variablen „widthOfCanvas" und „heightOfCanvas" zeigen den SchülerInnen, wie Variablen angelegt werden und wie Zuweisungen funktionieren. Das Zuweisen von Werten ohne neuerliche Benutzung des reservierten Wortes „var" wird in der Console des Browsers „live" vorgeführt. Dies soll den Lernenden die Funktionsweise der Console des Browsers zeigen. Falsche Bezeichnungen für Variablennamen finden sich in den Kommentaren in Abbildung 25 wieder und werden in der Console nach dem Auskommentieren als Fehler angezeigt. Somit sehen die SchülerInnen, wie Fehlermeldungen aussehen. Die verschiedenen Datentypen werden durch den Einsatz des typeof Operators und der Console erklärt. Dadurch lernen die SchülerInnen die Abfrage vom Datentyp einer Variablen und die Ausgabe eines Wertes durch die Console. Im nächsten Beispiel, Abbildung 26, werden arithmetische Operatoren vorgezeigt, deren Bekanntheit bei den SchülerInnen aufgrund des Mathematikunterrichts vorhanden ist.

```
<script>
    var widthOfCanvas = 1000;
    var heightOfCanvas = 800;

    var xPosRect = 400;
    var yPosRect = 150;
    var widthRect = 200;
    var heightRect = widthRect / 2;

    var xPosEll = xPosRect + 20;
    var yPosEll = yPosRect - 20;
    var widthEll = 100;
    var heightEll = 100;

    var counter = 0;

    function setup() {
        createCanvas(widthOfCanvas, heightOfCanvas);
        background(220, 220, 220);
    }

    function draw() {
        rect(xPosRect, yPosRect, widthRect, heightRect);
        ellipse(xPosEll, yPosEll, widthEll, heightEll);
        counter += 1; // -=, /=, *=
        console.log(counter);
    }
</script>
```

Abbildung 26: Beispiel arithmetische Operatoren

Es wird gezeigt, wie Variablen benutzt werden und wie mit ihnen durch Verknüpfung mit mathematischen Operatoren Berechnungen durchgeführt werden. Es wird in der Ausgabe das Überlappen des zuletzt gezeichneten Elements über alle anderen zuvor gezeichneten Elemente deutlich. Das schafft bei den Lernenden Verständnis für die Anordnungen von grafischen Anweisungen mit der Bibliothek p5.js.

Die „Counter" Variable zum Schluss in der Abbildung 26 in der Funktion draw beschreibt den kombinierten Zuweisungsoperator und zeigt optionalen Varianten der Operatoren. Die Ausgabe des Beispiels aus Abbildung 26 ist der Abbildung 27 ähnlich, allerdings mit dem Unterschied, dass die Ellipse weiß ist:

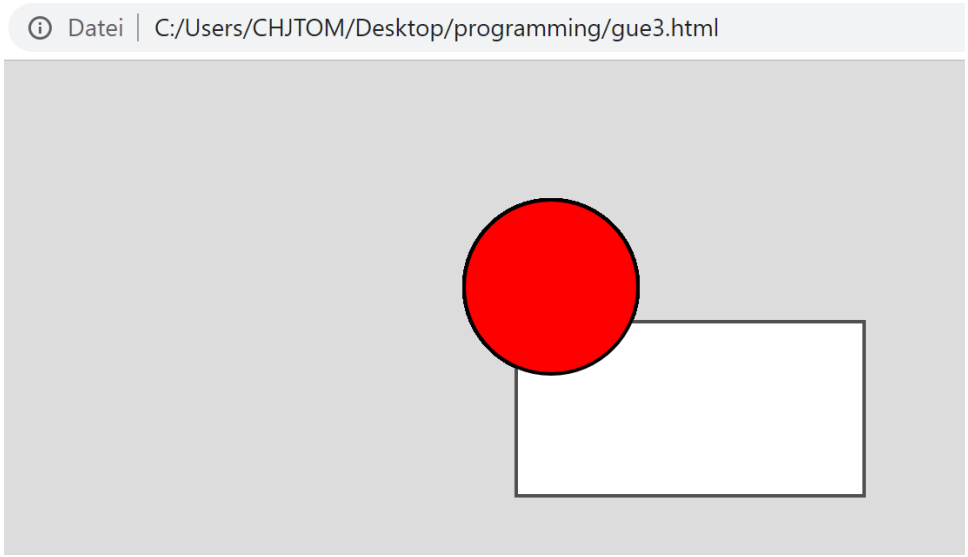

Abbildung 27: Ausgabe zu Beispiel 3

Das nächste Beispiel, Abbildung 28, zeigt das Verwenden von Farbe in p5.js. Die SchülerInnen lernen, wie Zeichenobjekte mit Farbe ausgefüllt werden und wie sie die Linien verändern können. Die Funktion draw aus dem vorigen Beispiel wird angepasst und liefert die Ausgabe aus Abbildung 27.

```
38  function draw() {
39      fill(255,255,255);
40      strokeWeight(2);
41      stroke(80);
42      rect(xPosRect, yPosRect, widthRect, heightRect);
43
44      stroke(0);
45      fill(255,0,0);
46      ellipse(xPosEll, yPosEll, widthEll, heightEll);
47      counter += 1; // -=, /=, *=
48      console.log(counter);
49  }
```

Abbildung 28: draw-Methode aus p5.js

Die Schwierigkeit für die SchülerInnen beim Ausfüllen von Objekten mit Farbe ist, dass die draw Funktion immer wieder wiederholt wird. Ist die fill-Color auf eine Farbe gesetzt, werden alle Elemente mit dieser Farbe ausgefüllt, auch wenn sie beim ersten Mal eine andere Farbe (weiß) besitzen.

Das letzte Beispiel, gezeigt in Abbildung 29, handelt von Vergleichsoperatoren. Diese Vergleichsoperatoren sind für den weiteren Kontrollfluss von Programmen relevant und werden durch eine Visualisierung von Text ausgegeben. Dadurch wird die Textausgabe in p5.js gelernt.

```
17          <script>
18              var widthOfCanvas = 1000;
19              var heightOfCanvas = 800;
20
21              function setup() {
22                  createCanvas(widthOfCanvas, heightOfCanvas);
23                  background(220, 220, 220);
24              }
25
26              function draw() {
27                  fill(255,255,255);
28                  textSize(30);
29                  var result;
30                  result = 1 == 1;
31                  text(result, 40, 40);
32                  result = 1 == "1";
33                  text(result, 40, 80);
34                  result = 1 === 1;
35                  text(result, 40, 120);
36                  result = 1 === "1";
37                  text(result, 40, 160);
38                  //<=, >=, <, >, !=, !==
39              }
40          </script>
```

Abbildung 29: Vergleichsoperatoren (grafisch)

Die verschiedenen Vergleichsoperatoren werden in dem obigen Beispiel, Abbildung 34, gezeigt und die Unterschiede hervorgehoben. Die „text" Funktion zur Ausgabe von „grafischem" Text wird in Abbildung 34 ausgegeben. Als letztes werden in einer separaten Übung der + Operator in einer anderen Funktion mit Strings in Verbindung gezeigt und Konvertierungen vorgenommen.

Abbildung 30: Ausgabe Vergleichsoperatoren (grafisch)

Der Unterschied zwischen den textuellen und visuellen Ausgaben wird hier deutlich. Selbst die grafische Repräsentation von Text in p5.js wirkt moderner und dynamischer als die Ausgabe von Text in der Console. Die textuelle Ausgabe in der Console wirkt sehr abstrakt und die Tendenz zu demotiviertem Lernen ist gegeben. Die Ausgaben der Beispiele

werden hier gezeigt, um den Unterschied zwischen den beiden Methoden klar hervorzuheben.

3.5.1.5 Übungen

Die Übungen für die SchülerInnen sind für das textuelle und visuelle Programmieren eigens erstellt. Die textuellen Übungen variieren aufgrund ihrer Beschaffenheit. Der Inhalt der Übungen deckt den Inhalt des Gelernten ab und auf dieselben Übungsinhalte zwischen grafisch und textuell wird geachtet. Die Übungen starten meist in den Unterrichtseinheiten sind aber so angelegt, dass es kaum möglich ist, diese innerhalb der Unterrichtsstunde fertigzustellen. Mit dem Rest der Übungen beschäftigen sich die SchülerInnen bis zur nächsten Stunde. Dadurch ist ein fortwährendes Auseinandersetzen bei den SchülerInnen gewährleistet ohne dabei explizit Hausübungen zu geben. Ein Auszug der Übungen passend zur ersten Unterrichtseinheit kann wie folgt gezeigt werden:

- Legen Sie Variablen an und bestimmen Sie Ihren Datentyp
- Benutzen Sie den Vergleichsoperator, um zwei Werte zu vergleichen

3.5.1.6 Kontrollfragen

Ein Auszug der Kontrollfragen in diesem Bereich sind wie folgt: welche Datentypen gibt es, was ist eine Variable, was ist dynamisches Typisieren, was sind Datentypen, was ist der Vorteil bzw. Nachteil von Datentypen und vieles mehr.

3.5.1.7 Kompetenzen

Die erreichten Kompetenzen passend zu unserem kompetenzorientierten Lehrplan der Oberstufen der Gymnasien im Wahlpflichtfach Informatik sind: Erkennen und Verwenden von Datentypen, Erstellen und Benutzung von Variablen, Anwenden von dynamischer Typisierung.

3.5.2 Einheit 2

Die Einheit zwei beschäftigt sich mit der Steuerung von Programmen mit Hilfe des if-Statements oder die Wiederholung von Codestücken mit Hilfe von Schleifen. Je nach Zustand soll ein Programm ablaufen.

3.5.2.1 Theorie

Die Theorie beschäftigt sich mit den folgenden Punkten:

- Kontrollstrukturen
- Vergleiche
- Logische Verknüpfungen

- Schleifen

Kontrollstrukturen bieten die Möglichkeit, Abläufe zu modellieren. Als Einstieg in die Kontrollstrukturen werden die Vergleiche beschrieben. Mit Vergleichen werden unterschiedliche Codesequenzen eingeführt. Diese Codesequenzen sind durch geschwungene Klammern (curly Brackets) begrenzt. Die Verknüpfung mehrerer Vergleich-Statements zusammen mit dem logischen UND oder dem logischen ODER sind oft eine Herausforderung für SchülerInnen. Deshalb werden Ergebnisse der logischen Operatoren anhand von Wahrheitstabellen erklärt. An dieser Stelle werden die ersten Beispiele, gefolgt von Übungen zu Vergleichen, durchgeführt.

Der zweite Bereich in dieser Einheit sind Schleifen. Die Schwierigkeit bei Schleifen liegt in den Abbruchbedingungen. Eine hohe Anzahl von SchülerInnen erzeugen Endlosschleifen, wenn sie das erste Mal eine Schleife programmieren. Aus diesem Grund sei speziell auf Inkremente, Dekremente usw. zu achten. Die Varianten for, while, und do-while werden erklärt und die Unterschiede der einzelnen Schleifen werden hervorgehoben. Weiter bekommen die Begriffe Endlosschleife, Inkrement bzw. Dekrement, Abschlussbedingung als auch break eine eindeutige Erklärung.

Am Ende der Unterrichtseinheit werden Funktionen zur Strukturierung eingeführt. Mit Codeblöcken in Zusammenhang stehen Funktionen. Diese werden ebenfalls als Begrenzung der Gültigkeit des Codes eingeführt. Begriffe wie Parameter, Call-by-value und Call-by-reference werden erklärt.

3.5.2.2 Beispiele textuell

Beim ersten Beispiel in dieser Unterrichtseinheit werden die ersten Versuche zu vergleichenden Kontrollstrukturen vorgestellt. Diese Strukturen werden, wie in Abbildung 35 gezeigt, mit "if" eingeleitet, gefolgt von einer Prüfbedingung und einem auszuführenden Codeblock, begrenzt durch geschwungene Klammern. Es werden den SchülerInnen unterschiedliche Varianten der "if" Anweisungen gezeigt und “else if” wird als weitere Variante erwähnt und live in den untenstehenden Code eingebaut.

```
8   <script>
9       var x = 100;
10      if (x > 100) {
11          console.log("BIG");
12      }
13      if (x < 100) {
14          console.log("SMALL");
15      }
16      if (x == 100) {
17          console.log("EXACT");
18      }
19      x = x*2;
20      if (x > 100) {
21          console.log("BIG");
22      } else {
23          console.log("SMALL OR EXACT");
24      }
25  </script>
```

Abbildung 31: Kontrollfluss in Javascript

Als Erweiterung des Beispiels wird das switch-Statement in Abbildung 32 gezeigt. Dieses switch-Statement zeigt den SchülerInnen den Vorteil der kürzeren Schreibweise gegenüber mehreren verschachtelten if-Statements. Am Rande wird der ternäre Operator erwähnt.

```
26      switch(x) {
27          case 50: console.log(50); break;
28          case 100: console.log(100); break;
29          case 200: console.log(200); break;
30          default: console.log("ALL OTHERS"); break;
31      }
```

Abbildung 32: switch-Statement

Die Ausgabe des Beispiels ist für den obigen Teil wie folgt:

```
EXACT                                                    ue5.html:17
BIG                                                      ue5.html:21
200                                                      ue5.html:29
```

Abbildung 33: Ausgabe switch-Statement

Als nächstes Beispiel (Abbildung 34) werden logische Operatoren zur Verbindung mehrerer Vergleichsoperatoren gezeigt.

```
8   <script>
9       var x = 100;
10      var y = 200;
11      if (x >= 100 && y < 300) {
12          console.log("RESULT IS OK!");
13      }
14
15      if (x == 100 || y < 100) {
16          console.log("RESULT IS OK!");
17      }
18  </script>
```

Abbildung 34: logische Operatoren

Beim Verstehen der logischen Verknüpfungen werden die Wahrheitswerte als Ergebnis einer Verknüpfung dargestellt und ein Bezug zu den Wahrheitstabellen aus dem Theorieblock wird hergestellt. Dadurch haben die SchülerInnen weniger Schwierigkeiten das logische UND, bei dem beide Bedingungen zwingend erfüllt sein müssen, und das logische ODER, bei dem nur eine Bedingung erfüllt sein muss, um in die Ausführung des Codes zu gelangen, zu verstehen. Unterschiedliche Abwandlungen der x- und y-Werte aus Abbildung 39 zeigen unterschiedliche Ausgänge der if-Statements. Die Ausgabe sieht hier wie folgt aus:

```
RESULT IS OK!                    ue6.html:12
RESULT IS OK!                    ue6.html:16
```

Abbildung 35: Ausgabe (logische Operatoren)

Das nächste Beispiel aus Abbildung 36 handelt von Schleifen und der Möglichkeit, wiederholt Ausführung von Code zu erreichen. Angefangen wird mit den “while” und “do-while” Schleifen, welche das grundlegende Verständnis zu Schleifen stärken, gefolgt von der “for”-Schleife, welche als verkürzte Schreibweise einer Schleife mit Inkrement angesehen wird.

```
<script>
    var counter = 0;
    var maxVal = 10;

    while (counter < maxVal) {
        console.log(counter);
        counter = counter + 1 //counter++;
    }

    counter = 0;
    do {
        console.log(counter);
        counter++;
    } while (counter < 0);

    for (var i = 0; i < maxVal; i++) {
        console.log(counter)
    }
</script>
```

Abbildung 36: Schleifen in Javascript

Die while-Schleife wird im Gegensatz zur do-while Schleife von SchülerInnen schneller verstanden. Ein deutlicher Hinweis auf die sichergestellte einmalige Ausführung des Codes bei do-while im Gegensatz zur while-Schleife, bei der die Bedingung gleich am Anfang abgefragt wird, ist für ein schnelles Weiterkommen wichtig. Der Aufbau der Schleifen mit den Abbruchsbedinungen und Inkrementen oder Dekrementen wird beschrieben. Bei der for-Schleife wird die verkürzte, übersichtliche Form erklärt und die häufige Anwendung dieser Form der Schleife in der Industrie und Wirtschaft wird markiert. Die reservierten Wörter “continue” und “break” spielen beim Steuern der Schleife eine entscheidende Rolle und werden in diesem Zusammenhang erwähnt.

3.5.2.3 Beispiele visuell

Das erste Beispiel aus Abbildung 37 in dieser Lehreinheit zeigt das Verwenden von Vergleichsbedingungen. Im Zusammenhang mit der draw-Funktion von p5.js werden diese Bedingungen veranschaulicht und erzeugen Dynamik im Ablauf des Programms, was bei der grafischen Ausgabe von Objekten besser merkbar ist als bei der textuellen Ausgabe.

```
<script>
    var widthOfCanvas = 1000;
    var heightOfCanvas = 800;
    var xPosRect = 400;
    var yPosRect = 150;
    var widthRect = 200;
    var heightRect = widthRect / 2;

    var changeColorOnFrame = 200;
    var growSizeOnFrame = 400;

    function setup() {
        createCanvas(widthOfCanvas, heightOfCanvas);
        background(220, 220, 220);
    }

    function draw() {
        if (frameCount < changeColorOnFrame) {
            fill(255,255,255);
        } else {
            fill(0,0,0);
        }
        if (frameCount >= growSizeOnFrame) {
            widthRect = widthRect + 1;
        }
        rect(xPosRect, yPosRect, widthRect, heightRect);
    }
</script>
```

Abbildung 37: Kontrollfluss in p5.js

Ein gezeichnetes Viereck aus dem Code in Abbildung 37 wird ab einem bestimmten Frame von weiß auf schwarz gezeichnet und ab einem weiteren Frame wächst das Rechteck. Veränderungen der Vergleichsbedingungen und die Folgerungen daraus zeigen den SchülerInnen das Anwendungsgebiet von if-Statements. Als nächstes Beispiel, zu sehen in Abbildung 38, werden logische Operatoren eingesetzt.

```
<script>
    var widthOfCanvas = 1000;
    var heightOfCanvas = 800;
    var xPosRect = 400;
    var yPosRect = 150;
    var widthRect = 200;
    var heightRect = widthRect / 2;

    var changeColorOnFrame = 200;
    var growSizeOnFrame = 400;
    var isWhite = false;

    function setup() {
        createCanvas(widthOfCanvas, heightOfCanvas);
        background(220, 220, 220);
    }

    function draw() {
        if (frameCount > 100 && frameCount < changeColorOnFrame) {
            fill(255,255,255);
            isWhite = true;
        } else {
            fill(0,0,0);
            isWhite = false;
        }
        if (frameCount > growSizeOnFrame || isWhite == false) {
            widthRect = widthRect + 1;
        }
        rect(xPosRect, yPosRect, widthRect, heightRect);
    }
</script>
```

Abbildung 38: logische Operatoren (grafisch) in p5.js

Für SchülerInnen ist die Funktion von isWhite im Zusammenhang mit der zweiten logischen Verknüpfung nicht leicht zu verstehen. Die erste logische Verknüpfung zwischen den Frames verstehen alle auf Anhieb sofort. Die Animation des wachsenden Rechtecks hat positives Feedback bei den Lernenden hervorgerufen. Das nächste Beispiel führt den SchülerInnen Schleifen vor. Dieses ist in Abbildung 39 zu sehen. Die drei Varianten while, do-while und for werden eingeführt und die Differenzen der Schleifen werden hervorgehoben:

```
<script>
    var widthOfCanvas = 1000;
    var heightOfCanvas = 800;
    var xPosRect = 0;
    var yPosRect = 0;
    var widthRect = 50;
    var heightRect = 50;
    var space = 10;

    function setup() {
        createCanvas(widthOfCanvas, heightOfCanvas);
        background(220, 220, 220);
    }

    function draw() {
        if (frameCount < 200) {
            fill(255,255,255);
            var i = 0;
            while (i < 5) {
                rect(xPosRect, yPosRect, widthRect, heightRect);
                xPosRect = xPosRect + widthRect + space;
                i++;
            }
        } else {
            var i = 0;
            do {
                fill(255,0,0);
                rect(xPosRect, yPosRect, widthRect, heightRect);
                xPosRect = xPosRect + widthRect + space;
                i++;
            } while (frameCount < 200 && i < 5);
        }
        xPosRect = 0;
    }
</script>
```

Abbildung 39: Schleifen (grafisch)

Die ersten beiden Varianten von Schleifen werden in Abbildung 40 beschrieben, die Unterschiede werden durch die Ausführung dargestellt. Die einmalige Ausführung der do – while Schleife wird in einem roten Viereck dargestellt. Die for-Schleife wird als Alternative zur while Schleife gezeigt (Abbildung 40). Dadurch sehen die Lernenden, dass eine Schleife in eine andere Schleifenvariante überführt werden kann.

```
for (var i = 0; i < 5; i++) {
    rect(xPosRect, yPosRect, widthRect, heightRect);
    xPosRect = xPosRect + widthRect + space;
}
```

Abbildung 40: for-Schleife in p5.js

Die verkürzte Form ist für die SchülerInnen meist ansprechender und die drei Teile, Initialisierung, Vergleichsbedingung und Inkrement, sind übersichtlich und leichter zu merken.

3.5.2.4 Übungen

Ein Auszug der gestellten Aufgaben im Bereich der Übungen ist:

- Schreiben Sie ein Programm, das je nach Variablenwert entscheidet, ob die Division fortgesetzt werden kann oder eine Division durch 0 vorliegt.
- Schreiben Sie ein Programm, das ab einem bestimmten Framecount ein weiteres Viereck hinzufügt.
- Schreiben Sie eine Schleife, welche die Zahlen von 0 – 100 ausgibt.
- Schreiben Sie eine Funktion, welche 10 Rechtecke erzeugt.

3.5.2.5 Kontrollfragen

Ein Teil der gestellten Kontrollfragen ist hier: was ist eine Schleife, welche Arten von Schleifen gibt es, was ist eine Abbruchbedingung, was ist der Unterschied zwischen einer do-while und while Schleife.

3.5.2.6 Kompetenzen

Die erlangten Kompetenzen in der Einheit 2 sind: Arbeiten mit Schleifen, Verwenden von Abbruchbedingungen, Verwendung des Inkrementoperators bzw. Dekrementoperators, Erkennen und Verwenden der unterschiedlichen Arten von Schleifen

3.5.3 Einheit 3

Die Einheit 3 beschäftigt sich mit Funktionen und deren Anwendungen. Es wird ein kurzer Überblick zur prozeduralen Programmierung gegeben.

3.5.3.1 Theorie

In der Theorieeinheit dieser Lektion werden folgende Themen angesprochen: Funktionen, Methodennamen, Rückgabewerte, Parameter, prozedurale Programmierung, Scope und Code Guidelines werden erklärt. Die Einheit 3 hat als Ziel die Kenntnis über Funktionen zu vermitteln. Die Strukturierung von Code mittels Funktionen wird anhand eines Beispiels vorgezeigt. Begriffe wie Wartbarkeit, Effizienz oder Code Guidelines zur Einhaltung von Aussehen von Code werden besprochen. Eine Zuweisung von Funktionen an Variablen als Alias oder Referenz auf die Funktion werden in einer Live-Demonstration gezeigt sowie die Gestaltung von Methodennnamen, Self Invoking Functions und Rückgabe von Werten kommen zur Beschreibung.

Der Theorieteil prozedurale Programmierung wird hier ausführlich geklärt. Die Vorteile der prozeduralen Programmierung in effizienten Algorithmen wird besprochen und ein Hinweis auf andere Programmiersprachen wird hier gegeben.

Im folgenden Schritt werden Übungen zu Funktionen gemacht. Als nächstes werden Scopes in der Programmierung eingeführt. Hier wird dem Schüler, der Schülerin local und global Scope durch ein Beispiel näher gebracht. Das Verständnis von local und global Scope ist für viele SchülerInnen nicht einfach. Das Überschreiben von Werten innerhalb unterschiedlicher Scopes gilt es unter anderem zu verstehen. Mit Übungen in diesem Bereich sollen Scopes verstanden werden. Am Ende gibt es eine Besprechung über die gelernten Inhalte, eine Wiederholung und Übungen für das weitere Lernen.

3.5.3.2 Beispiele textuell

In diesem Beispiel, dargestellt in Abbildung 41, zeigt die Lehrer den SchülerInnen, wie Funktionen geschrieben werden, wie die Namensgebung von Funktionen erfolgt und wie die richtige Struktur einer Funktion aufgezeigt wird.

```
 8        <script>
 9            function helloWorld() {
10                console.log(helloWorld);
11            }
12
13            function avg(num1, num2, num3) {
14                var sum = num1 + num2 + num3;
15                var result = sum / 3;
16                console.log(result);
17                return result;
18            }
19
20            function getBigger(num1, num2) {
21                if (num1 > num2) {
22                    return num1;
23                } else {
24                    return num2;
25                }
26            }
27
28            helloWorld();
29            avg(10,20,30);
30            getBigger(100,150);
31        </script>
```

Abbildung 41: Funktionen in Javascript

Danach werden die Funktionsaufrufe gestartet und es wird erklärt, wie die Zuweisung von Werten in Form von Parametern funktioniert. Die Übergabe von Werten an eine Funktion ist ein ernstzunehmender Punkt, der von den wenigsten SchülerInnen von Anfang an verstanden wird. Eine Darstellung der Übergabe der Werte an die Funktion ist meist hilfreich für das Verstehen des Ablaufs. Funktionen sind in Javascript wie Objekte und können somit auch an Variablen übergeben werden oder als Parameter an eine Funktion weitergereicht werden. Ein Hinweis auf die Speicherung einer Funktion in einer Variable wird den SchülerInnen gegeben. Zum Schluss wird das return-Statement geklärt. Der Aufruf einer Funktion reicht meist nicht, um den Wert aus einer Funktion zu bekommen. Das return-Statement gibt den Ausstiegspunkt einer Funktion an und die SchülerInnen sind gefordert zu verstehen, dass ein return-Statement ein Abbruch der laufenden Funktion ist und den Wert liefert, welcher nach dem return folgt.

```
 9            function getInput() {
10                var input = prompt("GET VALUE: ");
11                return input;
12            }
13
14            var value = getInput();
15            console.log(typeof value);
16            console.log(value);
17        </script>
```

Abbildung 42: Rückgabewerte von Funktionen

In der Ausführung im obigen zweiten Beispiel aus Abbildung 42 wird das Einlesen von Werten über “prompt” beschrieben und wie Werte aus Benutzereingaben verwendet werden. In diesem kurzen Beispiel einer Funktion mit Rückgabewert wird der Rückgabewert im Anschluss analysiert und die SchülerInnen können durch mehrfaches Aufrufen der Webseite erkennen, was der Datentyp der Rückgabe der Funktion „prompt” ist.

Im Beispiel drei dieser Einheit, abgebildet in nächster Grafik (Abbildung 43), werden local und global Scope geklärt. Der Scope wird sehr oft missverstanden und die SchülerInnen verwenden die Variablen dann falsch. Das Beispiel hilft den SchülerInnen, das Konzept hinter global und local zu verstehen.

```
<script>
    var gValue = 100;
    console.log("Global: " + gValue);

    function printValue() {
        var lValue = 1000;
        console.log("Local: " + lValue);
        console.log("Global: " + gValue);
        gValue = gValue / 2;
    }

    printValue();
    console.log("Local: " + lValue);
    console.log("Global: " + gValue);
</script>
```

Abbildung 43: lokale und globale Scopes in Javascript

Die Abänderung des Wertes gValue in der printValue Funktion und der Ausgabe der Werte an unterschiedlichen Stellen und der Versuch des Aufrufs der Variable lValue außerhalb des definierten Bereiches unterstützen die SchülerInnen beim Verstehen.

3.5.3.3 Beispiele visuell

Das erste Beispiel, Abbildung 44, zeigt die Einführung in die Definition von eigenen Funktionen. Bisher wurden die Funktionen von p5.js benutzt, somit sind die SchülerInnen mit dem Verwenden von Funktionen ein wenig vertraut.

```
<script>
    var widthOfCanvas = 1000;
    var heightOfCanvas = 800;
    var xPosRect = 10;
    var yPosRect = 100;
    var widthRect = 200;
    var heightRect;

    function setup() {
        createCanvas(widthOfCanvas, heightOfCanvas);
        background(220, 220, 220);
        heightRect = calculateHalfHeight(widthRect);
        outputValue("HELLO WORLD!");
    }

    function draw() {
        fill(255,0,0);
        stroke(2);
        rect(xPosRect, yPosRect, widthRect, heightRect);
    }

    function outputValue(output) {
        fill(255,255,255);
        textSize(30);
        text(output, 10, 40);
    }

    function calculateHalfHeight(width) {
        var result = width / 2;
        return result;
    }
</script>
```

Abbildung 44: Funktionen (grafisch) in p5.js

Der Aufruf von Funktionen muss den SchülerInnen allerdings noch nähergebracht werden. In diesem Zusammenhang entsteht die Frage, wo die Funktion draw und setup aufgerufen wird. P5.js ist designt, um diese Funktionen automatisch aufzurufen. Wie die richtige Namensgebung ist, wird erklärt und die Rückgabe von Werten aus den Funktionen wird besprochen. Die Berechnung der Höhe findet in dem Beispiel durch eine Funktion statt und die Ausgabe des Textes durch die Textfunktion wird auch in eine Funktion ausgelagert. Die SchülerInnen verstehen dabei, dass Funktionen nicht unbedingt etwas zurückgeben müssen, aber um Werte aus Funktionen zurückzubekommen, passiert dies meist mit return-Statements. In diesem Zusammenhang wird dasselbe Beispiel benutzt, um global und local Scope zu beschreiben. Die Benützung der Variablen außerhalb der Funktionen sind für alle globalen Variablen möglich. Die Variable result kann außerhalb der Funktion nicht aufgerufen werden und in einer live-Demonstration wird das Fehlen der Variable außerhalb gezeigt. Das Verändern globaler Variablen wird in der Funktion calculateHalfHeight gemacht. Bei den Aufrufen werden außerdem Parameter an die Funktionen übergeben. Das Verstehen der Übergabe von Werten und der Veränderungen der Variablennamen durch bei Übergabe als Parameter und der Zusammenhang mit lokalen Variablen ist für SchülerInnen eine Herausforderung und soll mit Übungen weiter verbessert werden.

3.5.3.4 Übungen

Beispiele für Übungen in dieser Einheit sind:

- Schreiben Sie eine Funktion, die den Mittelwert aus 4 Werten ermittelt.
- Schreiben Sie eine Funktion für das Addieren, Subtrahieren, Multiplizieren und Dividieren zweier Zahlen
- Schreiben Sie eine Funktion zur Erstellung eines Kreises
- Schreiben Sie eine Funktion zur Berechnung der halben Höhe eines Rechtecks

3.5.3.5 Kontrollfragen

Mögliche Kontrollfragen in dieser Einheit sind: was ist eine Funktion, was sind Parameter, wie wird ein Wert zurückgegeben, welche Vorteile haben Funktionen, was ist der Unterschied von global und local Scope, ...

3.5.3.6 Kompetenzen

Vermittelte Kompetenzen in der Einheit drei sind das Arbeiten mit Funktionen, die Übergabe von Parametern, das Returnieren von Werten aus einer Funktion, Anwenden von Funktionen zur Übersichtlichkeit und Wartbarkeit, Anwenden von local und global Scope.

3.5.4 Einheit 4

In der Einheit vier werden objektorientierte Konzepte beschrieben und die Programmierung mit Objekten in Javascript wird gezeigt.

3.5.4.1 Theorie

In der Theorieeinheit dieser Unterrichtsstunde werden Objekte erklärt und ihr Einsatzgebiet in der objektorientierten Programmierung wird beschrieben. Die Attribute (private, protected, public) werden in der objektorientierten Programmierung erklärt, Klassen werden besprochen, Instanzieren eines Objekts wird erklärt, abstrakte Klassen und die 4 Grundpfeiler der Programmierung werden ebenfalls beschrieben. Ein Auszug der vermittelten Inhalte sind in Abbildung 49 zu finden (Generalisierung, Vererbung, Kapselung und Polymorphismus).

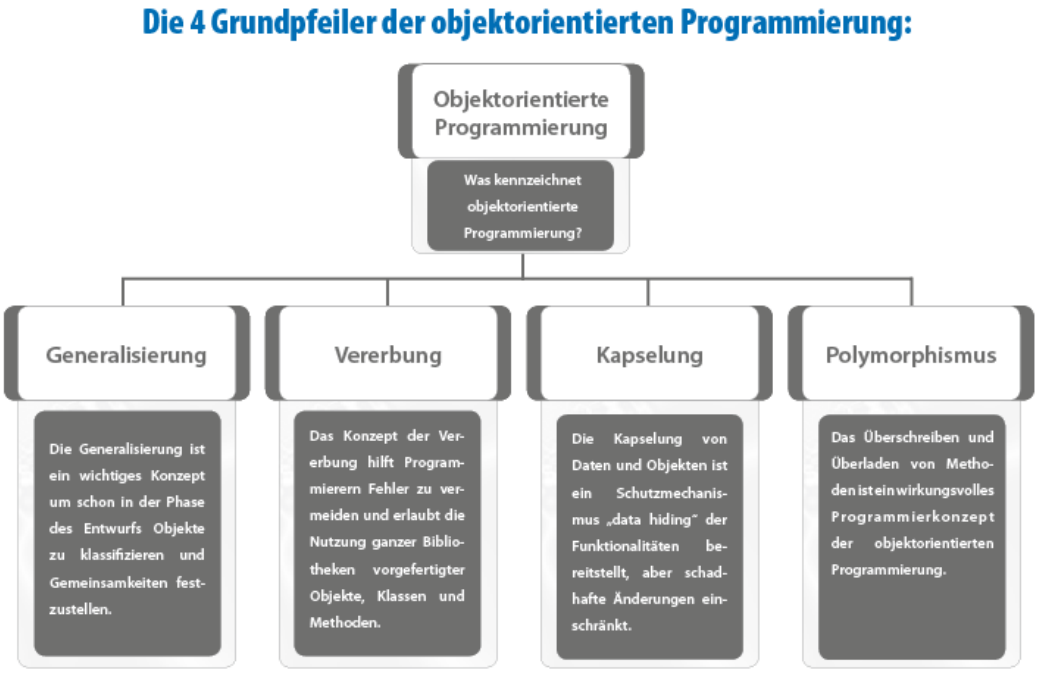

Abbildung 45: Theorie objektorientierte Programmierung [52]

Die Einführung von Objekten, welche Eigenschaften und Funktionen sie besitzen können und die Theorie hinter private, protected, public Eigenschaften von Variablen ist ein wichtiger Punkt, den es für SchülerInnen zu verstehen gilt. Die SchülerInnen lernen ebenfalls die Theorie zu Interfaces und Mehrfachvererbung bei objektorientierter Programmierung.

3.5.4.2 Beschreibung

Das Konzept von Klassen, abstrakten Klassen sowie der New Operator für das Anlegen neuer Objekte wird den SchülerInnen erklärt. Das Erstellen von Objekten und der Aufruf

von Attributen und Methoden durch die Punktnotation wird beschrieben und programmiert. Es werden die Unterschiede zwischen den Funktionen aus der letzten Einheit und den Objekten erklärt. Es wird auf prozedurale und objektorientierte Konzepte eingegangen und Unterschiede werden ermittelt [48].

Die SchülerInnen arbeiten mit Datentypen string, boolean und number als Objekte und lernen die Unterschiede kennen, zum Beispiel: Vergleiche mit Objekten sind anderes ist als Vergleiche mit primitiven Datentypen.

Der Ausdruck getter und setter wird erklärt und auf die Kapselung von Werten in Objekten wird hingewiesen, wenngleich auf Attribute in Javascript immer durch die Punktnotation zugegriffen werden kann. Das Löschen von Werten aus einem Objekt wird mittels dem Schlüsselwort delete beschrieben und die SchülerInnen führen Beispiele durch und lernen die Schleifen for-in und for-of. Der Umgang mit Vererbung wird gelehrt und prototype wird eingeführt. Ein Beispiel mit dem Anlegen von „Klassen", Objekten in Javascript, gezeigt in Abbildung 46, mit Attributen und Methoden, wird durchgenommen und mehrere Klassen leiten davon ab.

3.5.4.3 Beispiele textuell

In diesem Beispiel aus Abbildung 46 werden den SchülerInnen Objekte beigebracht. Die Objekte werden mit Attributen beschrieben, die auf unterschiedliche Art und Weise in Javascript erstellt werden können.

```
<script>
    var car = new Object();
    car["brand"] = 'Trabbi';
    car.color = 'green';
    car.ps = 30;
    car.drive = function () {
        console.log("CAR, GO FORWARD!");
    }

    var truck = {
        brand: "MAN",
        color: "silver",
        ps: 300,
        drive: function() {
            console.log("TRUCK, GO FORWARD!");
        }
    }

    function getPS() {
        console.log(this.ps);
    }

    car.getPS = getPS;
    truck.getPS = getPS;

    console.log(car.brand + " " + car["color"]);
    console.log(truck["brand"] + " " + truck.color);
    car.drive();
    truck.drive();
    car.getPS();
    truck.getPS();
</script>
```

Abbildung 46: Objekte in Javascript

Der New-Operator und das Instanzieren als Begriff werden eingeführt. Die Zuweisungen der Attribute und die dynamische Erweiterung der Attribute werden vorgeführt. Die Punktnotation oder die Zuweisung von Werten durch die eckige Klammer werden als Möglichkeit zur Wertesetzung von Attributen mit Objekten beschrieben. Als letztes werden Werte abgerufen. Die Aufrufe der Funktionen liefern die Werte aus dem Objekt und das Objekt als virtuelles Konstrukt zur Speicherung von Zuständen wird eingeführt. Das Keyword “this” ist im Zusammenhang mit Objekten für SchülerInnen zu Beginn herausfordernd. Aus diesem Grund wird ein weiteres Beispiel aus Abbildung 47durchgeführt.

```
<script>
    function Bike(weight, ps, brand) {
        this.weight = weight;
        this.ps = ps;
        this.brand = brand;
    }
    var myBike = new Bike(100, 5);
    console.log(myBike.ps);
    console.log(myBike.weight);
    console.log(myBike.brand);
    myBike.brand = "Ducati";
    console.log(myBike.brand);
</script>
```

Abbildung 47: Objekte erzeugen

In obigem Beispiel, Abbildung 47, wird eine Methode zum Erstellen von Objekten gezeigt. Diese Methode ähnelt sehr den Klassen von objektorientierten Programmiersprachen und ist deshalb für das vollständige Erklären und Verstehen von Objekten für SchülerInnen wichtig. Die Funktion Bike ist ein Konstruktor, mit dem mehrere Objekte Bike erstellt werden können. Das dritte Beispiel in Abbildung 48 beschäftigt sich mit der Vererbung, einem objektorientierten Konzept.

```
<script>
    function Parent(familyName) {
        this.familyName = familyName;
    }
    function Child() {

    }
    Child.prototype = new Parent('SAMPLE');
    var child = new Child();

    console.log(child.familyName);

    console.log(child.__proto__);
    console.log(Child.prototype);
</script>
```

Abbildung 48: Vererbung in Javascript

Die Vererbung wird durch prototype realisiert beziehungswiese durch das Attribut __proto__ von einem Objekt abgerufen. Der SchülerInnen sehen, wie der Wert vom übergeordneten Objekt an das untere Objekt ableitet und wie auf diese Werte zugegriffen wird. Eine Typabfrage ist durch das Attribut __proto__ möglich.

3.5.4.4 Beispiele visuell

Im ersten Beispiel, visualisiert in Abbildung 49, werden Objekte und Attribute erklärt. Der Aufbau der Übung ist so gestaltet, dass ein Objekt angelegt wird und Werte und Funktio-

nen als Attribute des Objektes definiert werden. Die Zeichenfunktion von p5.js wird umgeschrieben, sodass die Befehle zum Zeichnen ins Objekt wandern und von dort aufgerufen werden. Dadurch sehen die SchülerInnen die Funktion von Objekten. Die visuelle Ausgabe zeigt das virtuelle Objekt „gameObject“ den SchülerInnen grafisch repräsentiert im Browser und schafft Verständnis.

```
<script>
    var widthOfCanvas = 1000;
    var heightOfCanvas = 800;

    var gameObject = {
        widthRect: 200,
        heightRect: 100
    }
    gameObject['xPosRect'] = 10;
    gameObject['yPosRect'] = 100;
    gameObject.draw = function() {
        fill(255,0,0);
        stroke(2);
        rect(this.xPosRect, this.yPosRect, this.widthRect, this.heightRect);
    };

    function setup() {
        createCanvas(widthOfCanvas, heightOfCanvas);
        background(220, 220, 220);
    }

    function draw() {
        gameObject.draw();
    }
</script>
```

Abbildung 49: Objekte (grafisch)

Die Punktnotation und die Wertezuweisung mit eckigen Klammern werden gezeigt. In der nächsten Übung werden new als Operator zur Instanzierung und der Konstruktor eingeführt. Der Verweis auf Werte im Objekt selbst wird mit this gezeigt (Abbildung 50).

```
<script>
    var widthOfCanvas = 1000;
    var heightOfCanvas = 800;

    var Circle = function() {
        this.x = 100;
        this.y = 100;
        this.size = 50;
        this.draw = function() {
            ellipse(this.x, this.y, this.size, this.size);
        };
    };
    myCircle = new Circle();

    function setup() {
        createCanvas(widthOfCanvas, heightOfCanvas);
        background(220, 220, 220);
    }

    function draw() {
        myCircle.draw();
    }
</script>
```

Abbildung 50: this Keyword und Funktionsweise

Das letzte Beispiel, dargestellt in Abbildung 51, zeigt kurz das Arbeiten mit Vererbung. Das Attribut prototype und __proto__ werden eingeführt. Die SchülerInnen sehen, wie die Attribute x, y von „GameObject“ auf „Circle“ vererbt werden.

```
<script>
    var widthOfCanvas = 1000;
    var heightOfCanvas = 800;

    function GameObject() {
        this.x = 0;
        this.y = 0;
        this.size = 50;
    }

    function Circle() {
        this.draw = function() {
            fill(255, 0, 0);
            ellipse(this.x, this.y, this.size, this.size);
        };
    };
    Circle.prototype = new GameObject();

    myCircle = new Circle();
    myCircle.x = 100;
    myCircle.y = 100;

    function setup() {
        createCanvas(widthOfCanvas, heightOfCanvas);
        background(220, 220, 220);
    }

    function draw() {
        console.log(myCircle);
        myCircle.draw();
    }
</script>
```

Abbildung 51: Vererbung in p5.js

Das gameObject Objekt steht als übergeordnetes Objekt wie eine abstrakte Klasse dar.

3.5.4.5 Übungen

Beispielhafte Übungen in dieser Einheit sind wie folgt:

- Erstellen Sie ein Objekt Computer und geben Sie ihm Attribute, cpu, ram…
- Erstellen Sie ein Objekt Viereck. Die Attribute x, y und Größe sowie die draw Funktion soll im Objekt vorkommen.
- Erstellen Sie ein Objekt Roboter und definieren Sie passende Funktionen dazu: gehen, stehen, links, rechts...

3.5.4.6 Kontrollfragen

Ein Auszug der Kontrollfragen ist wie folgt: was ist ein Objekt, was ist die Kapselung von Werten, was ist Vererbung, was ist Generalisierung, was ist Polymorphismus, wie kann ein Objekt erzeugt werden?

3.5.4.7 Kompetenzen

Die vermittelten Kompetenzen in dieser Einheit sind: Erstellen und Arbeiten mit Objekten, Benutzung von Attributen in Objekten, Funktionen in Objekten und Vererbung und Kapselung von Werten in Objekten.

3.5.5 Einheit 5

In der Einheit 5 werden Listen und die Vorteile sowie assoziative Arrays zum namensbezogenen Referenzieren von Werten beschrieben.

3.5.5.1 Theorie

Der Fokus in dieser Einheit liegt bei Listen und Arrays. Es werden assoziative Arrays, Dictionaries, Indizes und Grundlegendes zu Such- und Sortieralgorithmen behandelt. Zu Beginn werden Listen und Arrays eingeführt. Es gibt einige Begriffe, welche in diesem Zusammenhang durchgenommen werden, zum Beispiel wird key-value mit der Verbindung zu Directory den SchülerInnen erklärt. Werte von Objekten können mittels brackets gesetzt werden. Die Ähnlichkeiten zwischen benannten Listen und Objekten in Javascript wird hervorgehoben. Der Index von Listen, gemischte Listen in Javascript als auch typisierte Listen, werden anschließend besprochen und Parallelen zu anderen Programmiersprachen werden gezeigt. Ein kurzer Hinweis auf Typescript als Abwandlung von Javascript wird gegeben. Bei den Indizes von Arrays wird auf Unterschiede zum Anfangswert 0, 1 in verschiedenen Programmiersprachen aufmerksam gemacht.

Die Speicherung von mehreren Werten in einer Variable mit Hilfe von Listen und weitere Vorzüge von Listen sowie vorprogrammierte Funktionen auf Arrays werden den SchülerInnen erklärt: length, pop, push, shift, unshift, valueof, join, splice, unsplice (Abbildung 53, Abbildung 54). Passend dazu wird der Begriff des Stacks eingeführt und das Konzept dahinter gezeigt. In den Beispielen werden Schleifen über Arrays durchgeführt und erste Schritte in Richtung Sortier- bzw Suchalgorithmen werden unternommen, zum Beispiel: Bubblesort (Abbildung 52).

3.5.5.2 Beispiele textuell

Im ersten Beispiel, Abbildung 52, dieser Einheit werden die Methoden zur Erzeugung eines Arrays gezeigt und welche unterschiedlichen Arten von Arrays es gibt.

```
<script>
    var myValues = [1, 200, 30, 54, 86, 79];
    var myOthers = new Array();
    myOthers[0] = 10;
    myOhters[1] = 55;
    myOthers[2] = 67;
    myOthers[3] = 78;
    myOthers[4] = 80;

    console.log("GET 3rd value of both: " + myValues[2] + " " + myOthers[2]);

    var person = [];
    person['name'] = "SAMPLE";
    person['age'] = 10;

    console.log("NAME: " + person['name'] + " AGE: " + person.age);

    var myMixed = ["NEWSAMPLE", 10, 190];
    console.log(myMixed[0]);
    console.log(myMixed[1]);
</script>
```

Abbildung 52: Sortieren und Suchen in Javascript

Es wird gezeigt, wie SchülerInnen auch mit new Array() Arrays erstellen können. Die einzelnen Werte werden dann mit Hilfe des Index zugewiesen. Zwei Fehlerquellen passieren SchülerInnen im Zusammenhang mit Indizes: der Start ist bei 0 und die Länge des Arrays (Anzahl der Elemente) ist größer als der letzte verfügbare Index. Assoziative Arrays (key-value pairs), einheitliche Arrays mit gleichen Datentypen und Arrays mit gemischten Datentypen werden erläutert.

```
<script>
    var myArr = [1, 2, 3, 4, 5, 6];
    myArr.push(100);
    console.log(myArr);
    console.log(myArr.length);
    myArr.pop();
    console.log(myArr);
    myArr.shift(100);
    console.log(myArr);
    myArr.unshift();
    console.log(myArr);
    console.log(myArr.valueOf(4));

    for (var i = 0; i < myArr.length; i++) {
        console.log(myArr[i]);
    }
</script>
```

Abbildung 53: Funktionen von Arrays in Javascript

Den SchülerInnen werden in diesem Beispiel die wichtigsten Funktionen zu Arrays und deren Bearbeitung und Abfrage erklärt. Eine Iteration mit Hilfe einer for-Schleife und der Funktion length auf Arrays wird vorgezeigt. pop und push als auch shift und unshift sowie join zum Zusammenfügen und Sortieren von Werten werden vorgezeigt. Splice und unsplice werden zur Vollständigkeit erwähnt.

```
<script>
    var myArr = [4, 3, 2, 1, 5, 6];
    myArr.push(100);
    console.log(myArr);
    console.log(myArr.length);
    myArr.pop();
    console.log(myArr);
    myArr.shift(100);
    console.log(myArr);
    myArr.unshift();
    console.log(myArr);
    console.log(myArr.valueOf(4));

    for (var i = 0; i < myArr.length; i++) {
        console.log(myArr[i]);
    }

    console.log(myArr.join(" "));
    console.log(myArr.sort());

    delete myArr[4];
    console.log(myArr);
    console.log(myArr.length);
</script>
```

Abbildung 54: Funktionen auf Arrays

3.5.5.3 Beispiele visuell

Das erste Beispiel, gezeigt in Abbildung 55, im Bereich Arrays beinhaltet die Definition und das Erzeugen von Arrays. Die Arrays werden eingesetzt, um mehrere Positionen von Vierecken zu bestimmen. Die visuelle Ausgabe hilft den SchülerInnen, die Abläufe von Arrays zu verstehen.

```
<script>
    var widthOfCanvas = 1000;
    var heightOfCanvas = 800;
    var xPositions = [50, 100, 150, 200];
    var yPosition = 100;
    var color = new Array();
    color[0] = 255;
    color[1] = 0;
    color[2] = 0;
    var backgroundColor = [];
    backgroundColor['red'] = 220;
    backgroundColor['green'] = 220;
    backgroundColor['blue'] = 220;
    var mixedArr = [100, 'rect'];
    console.log(mixedArr[0] + " " + mixedArr[1]);

    function setup() {
        createCanvas(widthOfCanvas, heightOfCanvas);
        background(
            backgroundColor['red'],
            backgroundColor['green'],
            backgroundColor['blue']
        );
    }

    function draw() {
        fill(color[0],color[1],color[2]);
        stroke(2);
        for (var i = 0; i < xPositions.length; i++) {
            rect(xPositions[i]+10, yPosition, 40, 40);
        }
    }
</script>
```

Abbildung 55: Arrays (grafisch)

Die Iteration durch ein Array mit vier Positionen lässt folgende Ausgabe, Abbildung 60, anzeigen.

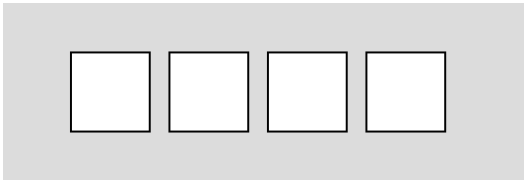

Abbildung 56: Visualisierung von Arrays in p5.js

Die unterschiedlichen Varianten der Erzeugung sind mittels new und Array Konstruktor oder der Erstellung mittels eckiger Klammern gegeben. Weiter werden assoziative Arrays, der Zugriff auf einzelne Werte und anschließend mixed Arrays, welche unterschiedliche Datentypen enthalten können, beschrieben. Im nächsten Beispiel, Abbildung 61, werden Funktionen auf Array vorgezeigt:

```
41          function draw() {
42              background(
43                  backgroundColor['red'],
44                  backgroundColor['green'],
45                  backgroundColor['blue']
46              );
47              showOutput();
48
49              fill(color[0], color.valueOf(1), color[2]);
50              stroke(2);
51              for (var i = 0; i < xPositions.length; i++) {
52                  console.log(i);
53                  rect(xPositions[i]+10, yPosition, 40, 40);
54              }
55              if (frameCount == 50) {
56                  output = "PUSH";
57                  xPositions.push(250);
58              }
59              if (frameCount == 100) {
60                  output = "POP";
61                  xPositions.pop();
62              }
63              if (frameCount == 150) {
64                  output = "UNSHIFT";
65                  xPositions.unshift(0);
66              }
67              if (frameCount == 200) {
68                  output = "SHIFT";
69                  xPositions.shift();
70              }
71          }
72
73          function showOutput(output) {
74              fill(255,255,255);
75              textSize(30);
76              text(output, 10, 40);
77          }
```

Abbildung 57: Funktionen auf Arrays (grafisch)

Die Ausgabe passiert dynamisch. Die Vierecke werden rechts und links zu Abbildung 58 dynamisch erweitert, um die Funktionen push, pop, unshift und shift zu simulieren. Die Differenz zwischen shift und pop muss deutlich herausgehoben werden.

3.5.5.4 Übungen

Einige der gestellten Aufgaben der Übungssequenzen sind hier zu finden:

- Sie haben ein Array mit 20 Werten gegeben. Schreiben Sie ein Programm, das alle Werte vom Array nimmt und von hinten nach vorne ausgibt.
- Schreiben Sie eine Funktion, die drei Werte an eine bestimmte Stelle im Programm einfügt.
- Legen Sie einen Array mit Objekten an. Jedes Objekt soll eine Person mit einigen Merkmalen als Attribute repräsentieren. Durchsuchen Sie das Array nach einem Objekt mit einem bestimmten Merkmal.
- Erstellen Sie ein Array mit 4 Vierecken. Fügen Sie vorne im Array 2 Vierecke in blau dazu und fügen Sie hinten 2 Vierecke in gelb dazu.

3.5.5.5 Kontrollfragen

Gestellte Kontrollfragen waren in diesem Kapitel folgende: was ist ein Array, was kann die Funktion pop und push, wozu dient die Funktion shift und unshift, was ist der Unterschied zwischen einem assoziativen Array und einem Objekt.

3.5.5.6 Kompetenzen

Die vermittelten Kompetenzen im Bereich der Arrays sind: das Anlegen von Arrays, das Einsetzen von Arrays, durch Arrays iterieren, assoziative Arrays, gemischte Arrays

3.5.6 Einheit 6

Die Einheit 6 beschäftigt sich mit dem Anwenden von Algorithmen, das Lesen von Pseudocode wird durchgenommen und eine kurze Einführung in die funktionale Programmierung wird gegeben.

3.5.6.1 Theorie

Die beiden Hauptteile in dieser Einheit sind: funktionale Programmierung und Algorithmen. Der größere Fokus liegt auf Algorithmen. Eine kurze Einführung in funktionale Programmierung wird gegeben. Für Zweiteres ist die Kompatibilität des gewählten Browsers von hoher Bedeutung. ECMAScript 5 ist Voraussetzung für den Einstieg in die funktionale Programmierung. ECMAScript 5-Funktionen wie map und reduce dienen als Vorführung. Eine Liste mit unterstützen Browsern lässt sich im Internet finden [51]. Es wird der Übergang der letzten Stunde auf das Thema Algorithmen als Lösungsrezepte für die Programmierung durchgeführt. Bei den Algorithmen sind einfache Sortieralgorithmen oder Suchalgorithmen für Einsteiger bedeutend.

Aus diesem Grund wird hier der Begriff des Algorithmus erläutert und einfache Sortieralgorithmen wie Insertionsort oder Selectionsort oder Suchalgorithmen wie lineare Suche theoretisch erklärt und dann wird ein Sortieralgorithmus praktisch anhand eines Beispiels (Abbildung 58) durchgeführt [50].

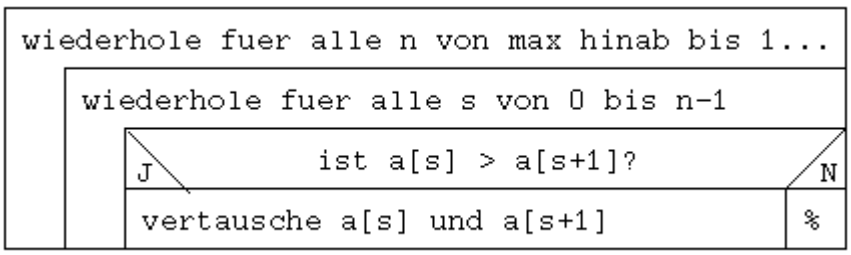

Abbildung 58: Pseudocode [18]

Alle Algorithmen werden als eine Art von Bauanleitung oder Kochrezept präsentiert, das Schritt für Schritt eine Lösung für ein Problem herbeiführt. Dabei wird gleich auf die Effizienz der Algorithmen Rücksicht genommen und eine kurze Einführung in die Bestimmung bzw. Klassifikation von Algorithmen wird gegeben. Bei der Übung zu Algorithmen werden in Beispielen einfache Lösungen für einfache Probleme gesucht.

Als nächster Punkt wird die funktionale Programmierung erklärt. Die Theorie hinter der funktionalen Programmierung wird einfach beschrieben und ein Beispiel wird gezeigt [49]. Die Übungen zur funktionalen Programmierung werden von den SchülerInnen durchgeführt.

3.5.6.2 Beispiele textuell

Das erste Beispiel in Abbildung 59 beschäftigt sich mit Sortieralgorithmen und zeigt den Ablauf von Bubblesort.

```
8   <script>
9       var myArr = [1, 6, 9, 20, 3, 15, 7, 3];
10      for (var i = myArr.length; i > 0; i--) {
11          for (var j = 0; j < i; j++) {
12              if (myArr[j] > myArr[j+1]) {
13                  var tmp = myArr[j];
14                  myArr[j] = myArr[j+1];
15                  myArr[j+1] = tmp;
16              }
17          }
18      }
19      console.log(myArr);
20  </script>
```

Abbildung 59: Implementierung von Bubblesort

Die Sortieralgorithmen spielen in der Informatik eine zentrale Rolle. Aus diesem Grund werden sie im Beispiel erklärt und vorgeführt. Die schrittweise Abarbeitung eines Prozederes, dem Bubblesort Algorithmus, wird beschrieben. Es wird den SchülerInnen das Anwenden von bewährten Algorithmen gezeigt. Die zweifachen Schleifen sind für SchülerInnen nicht einfach zu verstehen. Die Herausforderung besteht beim Umgang mit den Indizes. Der Ablauf des Algorithmus hat folgende Ausgabe:

```
▶ (8) [1, 6, 9, 3, 15, 7, 3, 20]                    ue17.html:18
▶ (8) [1, 6, 3, 9, 7, 3, 15, 20]                    ue17.html:18
▶ (8) [1, 3, 6, 7, 3, 9, 15, 20]                    ue17.html:18
▶ (8) [1, 3, 6, 3, 7, 9, 15, 20]                    ue17.html:18
▶ (8) [1, 3, 3, 6, 7, 9, 15, 20]                    ue17.html:18
▶ (8) [1, 3, 3, 6, 7, 9, 15, 20]                    ue17.html:18
▶ (8) [1, 3, 3, 6, 7, 9, 15, 20]                    ue17.html:18
▶ (8) [1, 3, 3, 6, 7, 9, 15, 20]                    ue17.html:18
▶ (8) [1, 3, 3, 6, 7, 9, 15, 20]                    ue17.html:20
```

Abbildung 60: Ausgabe Pseudocodeimplementierung

Das zweite Beispiel (Abbildung 61) in dieser Einheit beschäftigt sich mit funktionaler Programmierung. Funktionale Programmierung ist in verschiedenen Bereichen der IT von essentieller Bedeutung. Aus diesem Grund gibt es eine kurze Einführung und Erklärung. Dieser Teil ist bei beiden Teilgruppen (textuell und visuell) gleich, da die funktionale Programmierung nur als theoretischer Punkt für die SchülerInnen in dieser Unterrichtssequenz vermittelt wird. Das gezeigte Beispiel ist:

```
 8        <script>
 9            var closedDoor = {
10              open: false
11            };
12
13            function toggleDoor(door){
14              return { open: !door.open };
15            }
16
17            var openedDoor = toggleDoor(closedDoor);
18            console.log(openDoor);
19
20            var values = [1, 2, 3];
21
22            var squares = values.map(function square(value){
23              return value * value;
24            });
25            console.log(squares);
26        </script>
```

Abbildung 61: Funktionale Programmierung

Das Beispiel besteht aus zwei unterschiedlichen Bereichen. Der erste Bereich zeigt den Einsatz von Funktionen, welche nur Eingabe- und Ausgabewerte kennen müssen, um trotzdem „arbeiten" zu können. Der Kontext, die Abhängigkeit von globalen Werten, wird ausgeschlossen. Der zweite Bereich bearbeitet Werte einer Liste und erstellt die quadrierten Werte und retourniert diese in ein Array. Die Ausgabe des Beispiels ist:

```
▼ Object                                            ue16.html:18
    open: true
  ▶ __proto__: Object
▼ Array(3)                                          ue16.html:25
    0: 1
    1: 4
    2: 9
    length: 3
  ▶ __proto__: Array(0)
```

Abbildung 62: Ausgabe funktionale Programmierung

3.5.6.3 Beispiele visuell

Das erste Beispiel (Abbildung 63) beschäftigt sich mit dem Sortieralgorithmus Bubblesort. Für die Anzeige und die Sortierung mittels Bubblesort in p5.js müssen die SchülerInnen einige Punkte verstehen. Zum einen ist es eine komplexe Vorstellung, wie die Anzeige mit dem Sortieralgorithmus funktioniert, zum anderen ist der Sortieralgorithmus für die SchülerInnen nicht einfach zu verstehen, da er im Gegensatz zum Pseudocode bei der Umsetzung gesplittet werden muss. Die „draw" Funktion übernimmt in dem gezeigten Beispiel die Funktion der äußeren Schleife von Bubblesort und bestimmt somit den äußeren Index. Die Sortierung des Arrays, die innere Schleife, wird in sortArr() ausgeführt.

```
var widthOfCanvas = 1000;
var heightOfCanvas = 800;
var xPositions = [];
var yPosition = 40;
var myArr = [1, 6, 9, 20, 3, 15, 7, 3];
var sortPos = myArr.length;

function calculateXPositions(numOfElements) {
    var returner = [];
    for (var i = 0; i < numOfElements; i++) {
        returner.push(i*50);
    }
    return returner;
}

function setup() {
    createCanvas(widthOfCanvas, heightOfCanvas);
    background(220, 220, 220);
    rectMode(CENTER);
    xPositions = calculateXPositions(myArr.length);
}

function draw() {
    if (sortPos > 0) {
        sortArr();
        yPosition = yPosition + 50;
    }
    //background(220, 220, 220);
    fill(255, 0, 0);
    stroke(2);
    for (var i = 0; i < xPositions.length; i++) {
        rect(xPositions[i]+40, yPosition, 40, 40);
        fill(255,255,255);
        textSize(10);
        text(myArr[i], xPositions[i]+35, yPosition+5);
        fill(255,0,0);
    }
}
```

Abbildung 63: Sortieralgorithmus Bubblesort im p5.js Konstrukt

Hier ist die Programmierung der sortArr Funktion dargestellt (Abbildung 64):

```
function sortArr() {
    for (var i = 0; i < sortPos - 1; i++) {
        if (myArr[i] > myArr[i+1]) {
            var tmp = myArr[i];
            myArr[i] = myArr[i+1];
            myArr[i+1] = tmp;
        }
    }
    sortPos--;
}
```

Abbildung 64: Sortierfunktion Bubblesort

Die Ausgabe ist grafisch und präsentiert sich den SchülerInnen wie folgt:

Abbildung 65: Ausgabe Sortierung (grafisch)

Für SchülerInnen ist diese Ausgabe leichter zu verstehen als die textuelle Ausgabe.

3.5.6.4 Übungen

Ein Auszug der Übungen der Unterrichtsstunde wird hier gegeben:

- Bitte schreiben Sie ein Programm nach dem Pseudocode zum Sortieren mittels Insertionsort.
- Bitte schreiben Sie ein Programm nach dem Pseudocode für Selectionsort.
- Bitte schreiben Sie ein Programm nach dem Pseudocode zum Suchen mittels linearer Suche.

3.5.6.5 Kontrollfragen

Die gestellten Kontrollfragen in diesem Bereich sind: was ist ein Algorithmus, womit kann ein Algorithmus verglichen werden, was sind rekursive Funktionen, welche Sortier- und Suchalgorithmen sind bekannt, was ist Pseudocode?

3.5.6.6 Kompetenzen

Die vermittelten Kompetenzen in diesem Bereich sind: das Lesen von Algorithmen in Pseudocode, Schreiben von Algorithmen als Vorgangsweisen, das Schreiben von Sortier- und Suchalgorithmen, Verstehen von funktionaler Programmierweise und Verwenden von verschachtelten Funktionen und Schleifen.

3.5.7 Praktische Aufgabe

Die letzte Einheit dient als Wiederholung des Gelernten. Eine Übersicht über die einzelnen Bereiche, welche gelernt wurden, wird gegeben. Die SchülerInnen befassen sich mit der Lösung eines Problems in Form eines kleinen Programms. Vorbereitete Aufgaben in textueller Form, wie sie bei einer Anforderungs- bzw. Spezifikationsanalyse entstehen, werden den SchülerInnen vorgelegt. Dadurch sollen sie die fachliche Fähigkeit erlangen, aus einer Spezifikation einen ausführbaren Code zu schreiben. Die Konstrukte und die Art der Programmierung wählen die SchülerInnen selbst. Durch dieses Vorgehen entsteht die Kenntnis für das Anwenden des Gelernten. Die SchülerInnen starten mit der Programmierung und haben eine Woche Zeit, die Lösung für das gestellte Problem zu finden. Es ist nötig, in dieser praktischen Woche für etwaige Fragen der SchülerInnen zur Verfügung zu stehen.

3.5.8 Präsentation

In dieser Einheit findet die Überprüfung des Programmierten statt. Es werden die Programme der SchülerInnen vorgeführt und der Aufbau des Programmes wird von jedem Schüler und jeder Schülerin erklärt. Eine zentrale Komponente ist die Klärung des Lösungswegs. Die Frage, die sich hier stellt lautet: wie denkt der Schüler bzw. die Schülerin über seine bzw. ihre Lösung? Der Lösungsweg sollte anhand einer Präsentation oder des Programmes vorgeführt werden. Ein Teil der Lösung ist die Dokumentation des geschriebenen Codes. SchülerInnen sollen lernen, dass Programmcode dokumentiert wird, um die Zusammenarbeit bzw. Wartbarkeit des Codes zu erhöhen. Deshalb sollte auf diesen Punkt besonders Rücksicht genommen werden.

4 Testung

Die Entwicklung eines geeigneten Tests zur Überprüfung der Fähigkeiten und Kompetenzen der SchülerInnen war eine zeitintensive, komplexe Aufgabe. Es wurde auf alle vermittelten Lehrinhalte in den Lehreinheiten Rücksicht genommen. Die Unterrichtssequenzen mit den Lehrinhalten basieren auf den aktuellen Lehrplänen für das Wahlpflichtfach Informatik an der Oberstufe der Gymnasien.

4.1 Evaluierung der Unterrichtsmethode

Die textbasierten und visuellen Ausgaben der im Unterricht durchgeführten Programmierbeispiele werden durch quantitative und qualitative Methoden evaluiert. . Aus diesem Grund werden die Methoden kurz betrachtet und für die Testung der Programmiergruppen, Kontrollgruppe und Testgruppe, herangezogen. Das Ziel der Visualisierung der Ausgabe von geschriebenen Anweisungen, Codes, ist die Schaffung verständlicher, effektiver, leicht zu merkender Darstellungen zum effektiveren Lernen des Programmierens. Genau das, der Einfluss unterschiedlicher Ausgaben auf den Lernerfolg, wird in der Evaluierung geprüft.

4.2 Quantitative Evaluierung

Eine quantitative Analyse von Softwarevisualisierung misst die Eigenschaften der Visualisierung, zum Beispiel: Wieviel hat sich der Betrachter gemerkt? Ist der Ablauf der Visualisierung richtig verlaufen? Wieviel Information ist zu sehen? Und vieles darüber hinaus.

Die quantitative Evaluierung beschäftigt sich mit einer statistischen Analyse eines kontrollierten Experiments. Das Experiment wird durch Variablen und Faktoren beschrieben und kann deshalb messbar bewertet werden. Die unabhängigen Faktoren sind jene, welche durch die Personen bestimmt werden, die das Experiment erstellen. Die abhängigen Faktoren sind die aufgezeichneten Werte während des Experiments und die Antworten auf gestellte Fragen zum Experiment [24]. Kovariate Faktoren sind das Alter und das Vorwissen. Diese Faktoren beeinflussen das Ergebnis des Experiments.

Die statistische Analyse bezieht sich meist auf beschreibende und folgernde Statistik. Die typischen Kenngrößen sind der Median, das Mittel, die Verteilung, die Standardabwei-

chung und die Varianz. Die folgernde Statistik teilt meist in zwei Gruppen ein und führt einen T-Test auf beide Gruppen aus und versucht Signifikanz abzuleiten mit einer Fehlerwahrscheinlichkeit von 5% [24]. Voraussetzung für die Anwendung eines T-Test ist das Vorliegen einer Normalverteilung und gleiche Varianz bei beiden Stichproben. Ist die Varianz beider Testgruppen verschieden kommt der Welch-Test zum Einsatz. Die Normalverteilung der Stichproben wird mittels Kolgomorov-Smirnov Test oder Shapiro-Wilk Test geprüft. Für ein signifikantes Ergebnis wird bei quantitativen Evaluierungen eine ausreichende Anzahl von Testpersonen benötigt.

4.2.1 Evaluierung des Gelernten

Die Evaluierung des Gelernten hat zum Ziel, die Effektivität und den Lernprozess zu ermitteln. Eine Studie ermittelt die Effektivität des Gelernten, indem zwei Entscheidungen vorab getroffen werden: Welches Experiment wird durchgeführt und wer werden die Testpersonen sein? Die Testpersonen sollen der Zielgruppe angehören, für welche das Experiment durchgeführt wird. Oft gibt es nicht genug Testpersonen im Zielgruppenbereich und deshalb werden die Studien oft mit StudentInnen, meist aus dem psychologischen oder computerwissenschaftlichen Bereich durchgeführt. Die Testpersonen werden in zwei oder mehr Gruppen geteilt, welche unterschiedliche Methoden zur Erlernung des Algorithmus bzw. der Visualisierung bekommen [24]. Die Zuweisung passiert meist zufällig, wobei darauf geachtet werden sollte, dass die Fähigkeiten in den Gruppen gleichmäßig verteilt sind.

Die Bestimmung der Anzahl der Testgruppen hängt von den gewählten abhängigen und unabhängigen Faktoren ab. N unabhängige Variablen und M Ausprägungen pro Variable benötigen N x M Gruppen um eine signifikante Aussage zu treffen. Die meisten Studien beziehen sich auf eine oder zwei unabhängige Variablen [24]. In dieser Gruppe handelt es sich um zwei Testgruppen, da auf die Lerneffizienz bei unterschiedlicher Ausgabe (textuell und visuell) getestet wird.

Die Messung der Lerneffizienz erfordert geeignete Kenngrößen. Eine Kenngröße ist die Leistung während des Experiments durch Aufzeichnung des Verhaltens der StudentInnen. Sollte eine Aufzeichnung nicht möglich sein, können Tests danach ausgeführt werden, indem die Testpersonen nach dem Experiment eine Reihe von Fragen beantworten [24]. Der Sinn dieser Testungen danach ist, die Wahrnehmung und die Beschreibung über die getestete Sache (Konzept und Ablauf/Prozedere) zu überprüfen – hat die Testperson verstanden worum es geht. Mit einer Testung davor und einer Testung danach kann eine Entwicklung der getesteten Person überprüft werden.

Die Frage ist: funktioniert die Visualisierung als Lernunterstützung bzw. lernt der Lernende durch Beanspruchung beider Gehirnhälften durch die Visualisierung der Ausgabe wirklich am besten? Bei Algorithmusvisualisierungen wurde festgestellt, dass der Umgang mit der Visualisierung wichtiger ist als die Visualisierung selbst bzw. die Mischung aus Visualisierung und der Umgang mit der Visualisierung [27]:

"In particular, algorithm visualizations were efficient in scenarios were students had to actively solve prediction and programming exercises" [24].

Die Aussage ist eine der Grundlagen für die Entwicklung der vorgestellten Methodik zur Effizienzsteigerung des Programmierens in Oberstufen an Gymnasien.

4.3 Beschreibung

Der Ablauf der Testung basiert auf zwei Teilen, einem Theorieteil und einem Praxisteil. Der Theorieteil deckt alle Informationen ab, welche den SchülerInnen während der Lehreinheiten vermittelt wurden. Die Vorbereitung der SchülerInnen war in den Lehreinheiten durch Wiederholungen am Beginn der Stunde und der Kontrollfragen am Ende der Stunde gegeben. Der Theorieteil besteht aus Fragen basierend auf den vermittelten Lehrinhalten.

Der Praxisteil besteht aus einer Analyse eines vorgegebenen Codes und zwei praktischen Beispielen. Für die Testung sind zwei Stunden à 50 Minuten geplant. Bei der praktischen Prüfung werden Beispiele von den Unterrichtssequenzen aus den vergangenen Stunden abgeleitet und den SchülerInnen zur Unterstützung gegeben. Der Grund für einen theoretischen und praktischen Teil in der Prüfung ist jener, dass die Tätigkeit des Programmierens theoretische und praktische Teile beinhaltet. Programmieren selbst ist nicht die Beherrschung einer Programmiersprache. Es ist vielmehr eine Denkweise gemischt mit den Fähigkeiten, eine Programmiersprache benutzen zu können. Logisches Denken, Strukturierung und Wissen über Mögliches und Unmögliches in der Informatik machen das Programmieren aus.

Der konkrete Aufbau des Tests kann in unterschiedliche Bereiche gegliedert werden. Diese werden in der Auswertung und Analyse getrennt voneinander betrachtet und zu einem Gesamtergebnis konsolidiert. Der Fragebogen besteht aus den folgenden Bereichen:

- Anweisungen und Variablen
- Kontrollstrukturen
- Funktionen

- Listen
- Objekte
- Algorithmen

Der praktische Teil besteht aus zwei Bereichen:

- Codeanalyse
- Pseudocode
- Lösen zweier Programmieraufgaben

Bei der Codeanalyse wurden den SchülerInnen einige Codes zur Durchsicht gegeben und sie mussten erklären, was der Code verursacht und was das Ergebnis des Codes ist. Diese Fragestellung ähnelt der Fragestellung in Algorithmen und Datenstrukturen an der TU Wien, wo Studenten Codes zur Analyse vorgezeigt bekommen und die Ergebnisse und Algorithmen interpretieren. Ebenfalls inspiriert von den Prüfungen zu Algorithmen und Datenstrukturen an der TU Wien ist die Übung zum Schreiben eines Pseudocodes. Als letzter Teil ist das Schreiben von Code wie in den Übungen vorgesehen. Bei den Aufgaben wurde auf den textuellen und visuellen Teil der Lehreinheiten Bezug genommen.

Am Experiment haben 32 Personen aufgeteilt auf zwei Klassen teilgenommen. 16 Personen nahmen am klassischen Unterricht Teil und 16 Personen nahmen am grafischen Unterricht teil. Die Vortestung ermittelte die Grundkenntnisse und basierend darauf wird die Interpretation der Ergebnisse vorgenommen. Die Notenskala in Prozent liegt für Nicht Genügend, Genügend, Befriedigend, Gut und Sehr Gut bei <50, >=50, >=65, >=80, >=90.

4.4 Auswertung

Die Auswertung der Testung erfolgt in mehreren Teilen. Zuerst werden die einzelnen theoretischen Kategorien ausgewertet und eine Folgerung auf den gesamten Theorieteil wird gezogen. Anschließend werden die praktischen Teile bewertet. Die einzelnen Teilbereiche werden analysiert und ein Resümee über den praktischen Teil wird gezogen. Zum Schluss wird die gesamte Testung konsolidiert betrachtet und Vergleiche zwischen dem „klassischen" und dem „grafischen" Programmieren werden gezogen.

4.4.1 Theorieteil

Der Theorieteil besteht aus den sechs abgeprüften Kategorien. Jede wird einzeln betrachtet. Die Reihenfolge der Kategorien wird hier vorgegeben:

1. Anweisungen und Variablen

2. Kontrollstrukturen
3. Funktionen
4. Listen
5. Objekte
6. Algorithmen

4.4.2 Theorie zusammenfassend

Das Gesamtergebnis der Testung aus Abbildung 66 ergibt für das „klassische" Programmieren eine nahezu glockenförmige Verteilung der Noten mit gleicher Anzahl von negativen und sehr guten Noten. Der Hauptteil der Testergebnisse liegt zwischen Gut und Genügend, wobei die zentralen Noten über Befriedigend verteilt sind.

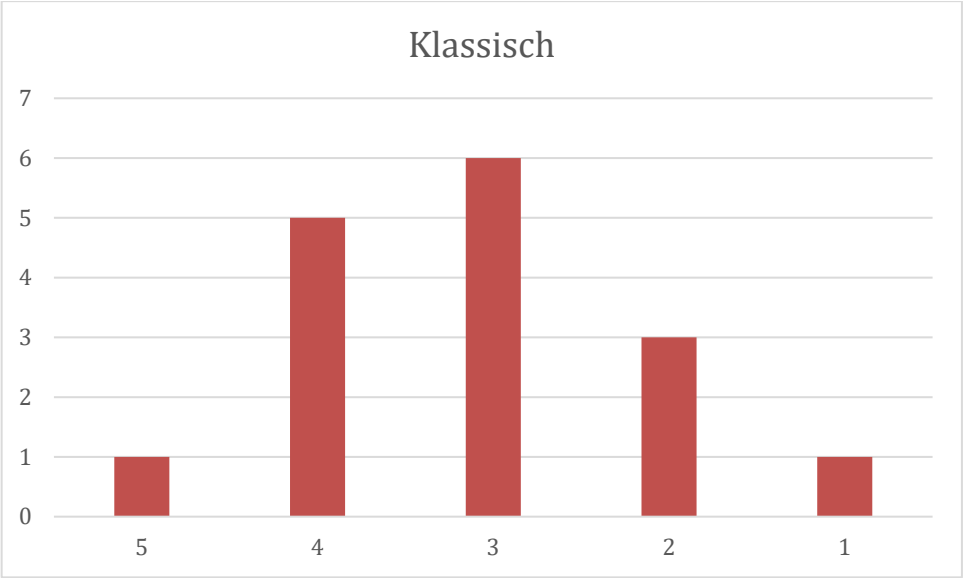

Abbildung 66: Notenverteilung der Theorie

Beim grafischen Programmieren konnten insgesamt folgende Ergebnisse erzielt werden. Sie sind in Abbildung 67 dargestellt.

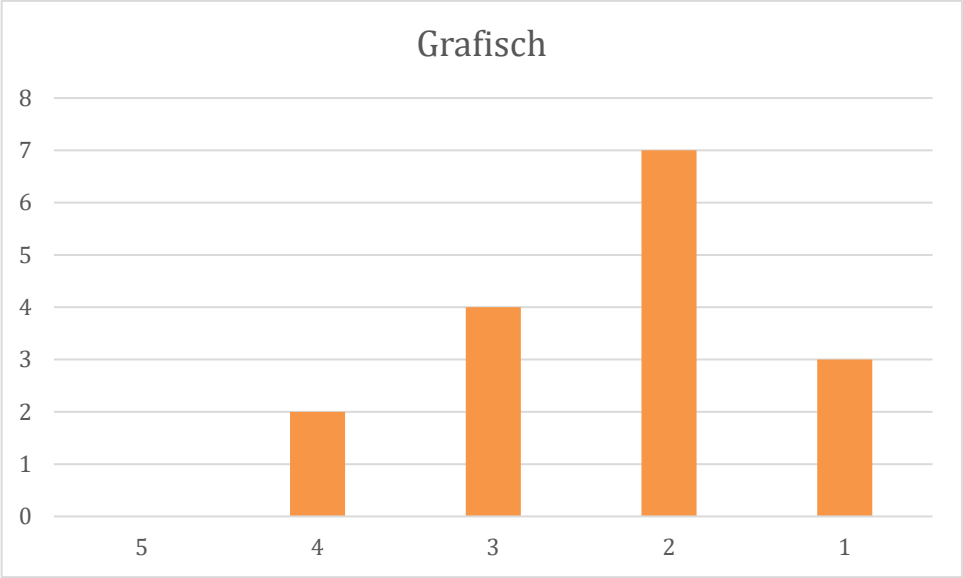

Abbildung 67: Notenverteilung der Theorie (grafisch)

Bei der Auswertung der Ergebnisse konnte folgende Notenverteilung bestimmt werden. Die glockenförmige Verteilung bei gleichem Test hat sich Richtung Gut und Sehr gut verschoben. Die meisten Noten finden sich bei Gut und Befriedigend. Negative Noten gibt es keine und die Anzahl der Sehr Gut ist mit drei mehr als bei zwei Genügend.

Der Vergleich der beiden Notenverteilungen „klassisch" und „grafisch" zeigt, dass es bei der „klassischen" Programmierung negative Noten gibt und es bei der „grafischen" Programmierung keine negative Note gibt. Die Anzahl der Genügend ist in beiden Verteilungen gleich. Die Befriedigend sind in der „grafischen" Programmierung weniger als in der „klassischen". Die Anzahl der Gut überwiegen stark bei der „grafischen" Programmierung. Die Sehr gut sind deutlich mehr bei der „grafischen" Programmierung als bei der „klassischen" Programmierung. Die glockenförmige Verteilung ist in der „grafischen" Programmierung nach rechts verschoben und zeugt von besseren Noten.

Im Verhältnis zueinander sind die Mittelwerte beider Notenverteilungen wie folgt: 2,81 „klassisch" und 2,38 „grafisch" mit einem Verhältnis von 1,35 zueinander, was bedeutet, dass die Methode der grafischen Programmierung eine 35 prozentige Steigerung der Noten bewirkte.

Weitere Ergebnisse der beschreibenden Statistik lieferten folgende Informationen über beide Experimente. Im Boxplotdiagramm der Punkteverteilungen aus Abbildung 68 ist zu erkennen, dass die besseren Ergebnisse beim grafischen Programmieren erzielt wurden. Dies ist damit zu erklären, dass die Mehrheit der Ergebnisse beim „grafischen" Program-

mieren zwischen 46 und 51 Punkten liegen und beim „klassischen“ Programmieren liegen die Ergebnisse zwischen 38 und 46 Punkten liegen.

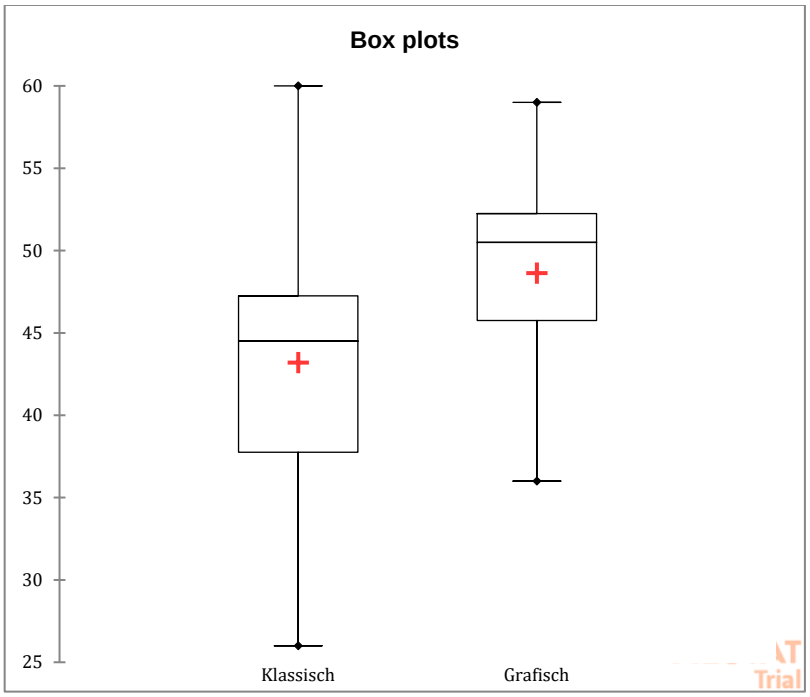

Abbildung 68: Boxplot der Punkteverteilung der Theorie

Die Minimum- und Maximumwerte liegen bei beiden Versuchsgruppen in etwa gleich. Die Mediane liegen zu Gunsten der „grafischen“ Gruppe um 3 Punkte auseinander.

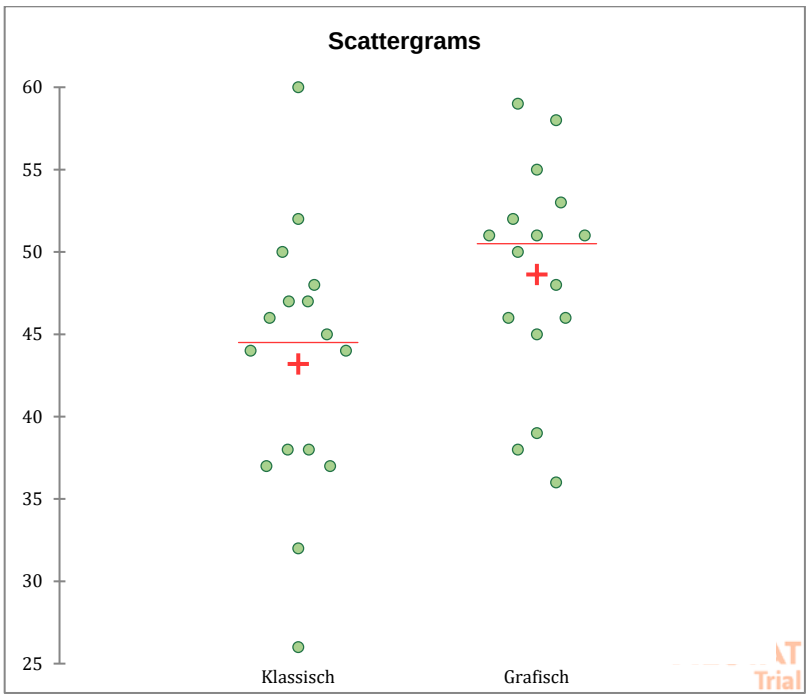

Abbildung 69: Punkteverteilung der Theorie

Die Verteilung der einzelnen Werte lassen sich grafisch wie oben in Abbildung 69 dargestellt zeigen. Die Werte sind bei der „grafischen“ Gruppe etwas höher angesiedelt als bei der „klassischen“ Gruppe. Bei der „klassischen“ Programmierung sind die Ergebnisse etwas dichter zwischen 43 und 50 Punkten angesiedelt. Besser kann das im folgenden Diagramm, Abbildung 70, dargestellt werden, wo die Ergebnisse beim „grafischen“ Programmieren gleichmäßig verteilt sind, aber eine Tendenz der Ergebnisse zwischen 45 und 60 Punkten liegt.

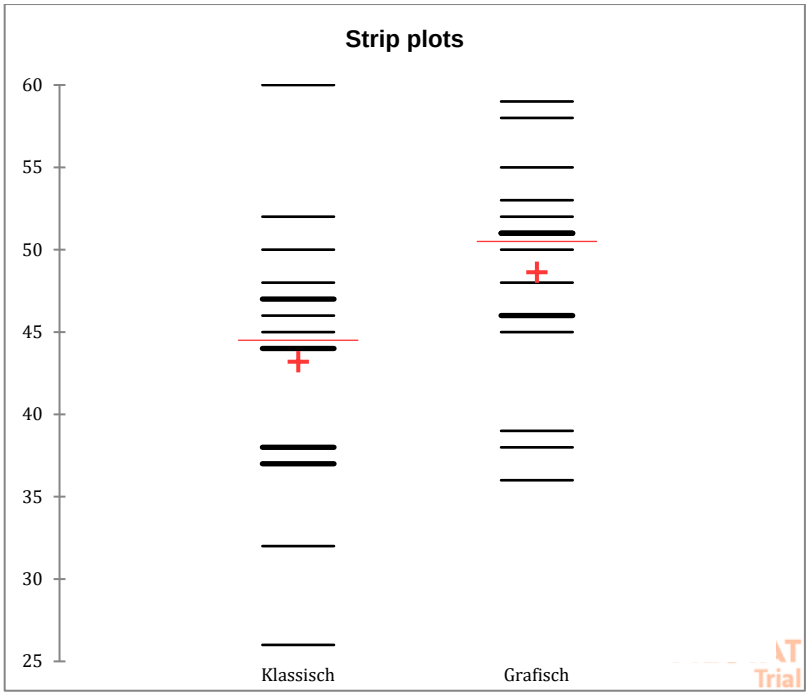

Abbildung 70: Intensität der Punkteverteilung der Theorie

In der „klassischen“ Gruppe gibt es zwei Ausreißer sowohl in positive als auch negative Richtung mit 60 und knapp 25 Punkten.

Die Verteilung der Stichproben ist mit dem Kolmogorov-Smirnov Test überprüft, welcher nach Ausführung auf den beiden Stichproben eine gleiche Verteilung beider Stichproben bei einem Signifikanzniveau von 1% liefert. Der Test zeigt, dass beide Stichproben normalverteilt sind. Die Verteilung kann wie folgt dargestellt werden:

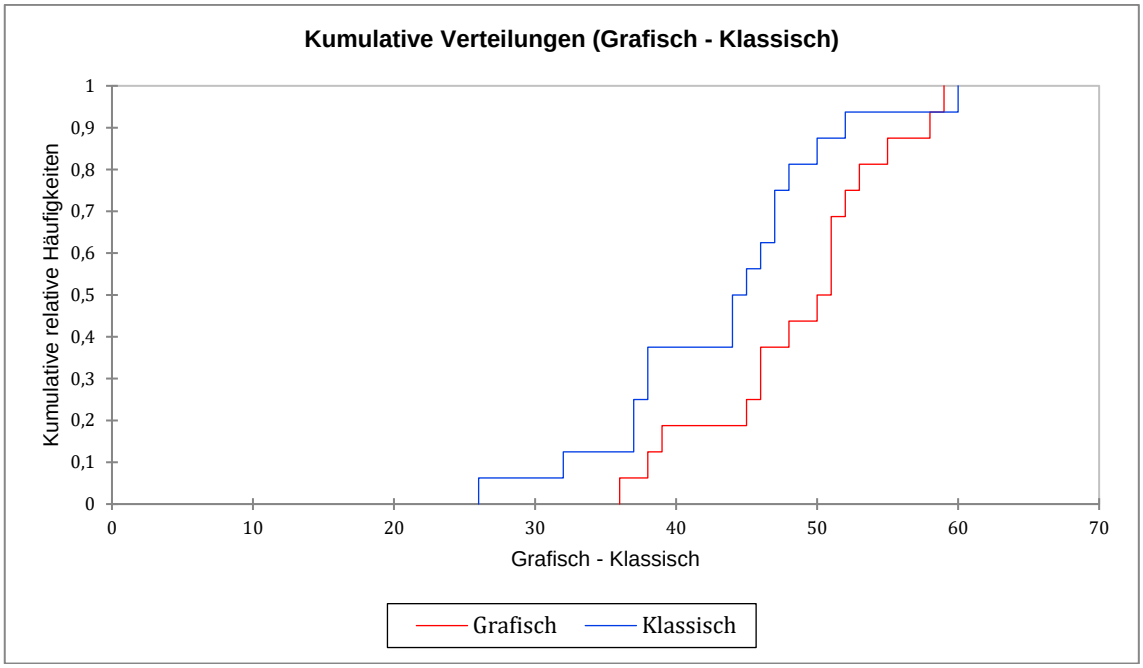

Abbildung 71: Vergleich der Stichproben - Grafisch - Klassisch

Durch die gleiche Verteilung beider Stichproben und dem logarithmischen Aussehen der Verteilungen wird eine Normalverteilung angenommen. Grafisch kann die Normalverteilung auch, wie folgt, dargestellt werden:

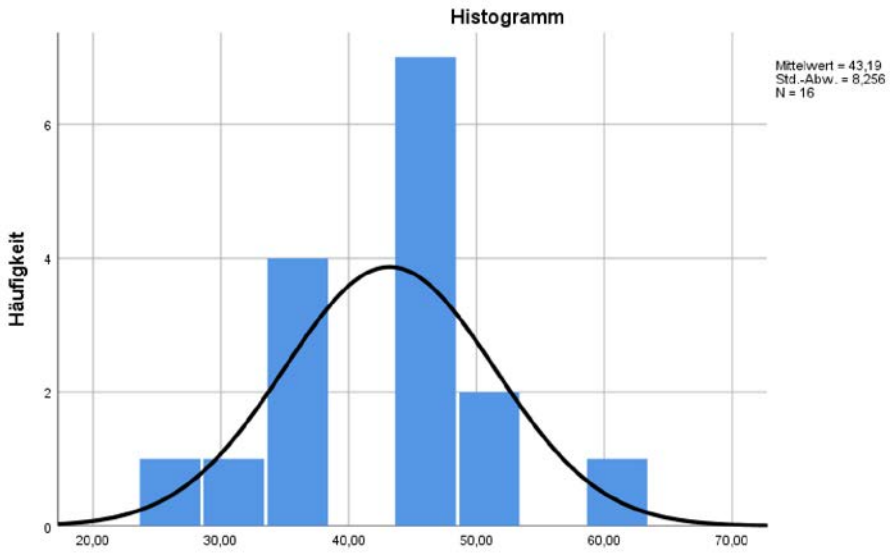

Abbildung 72: Verteilung der Ergebnisse (klassische Gruppe)

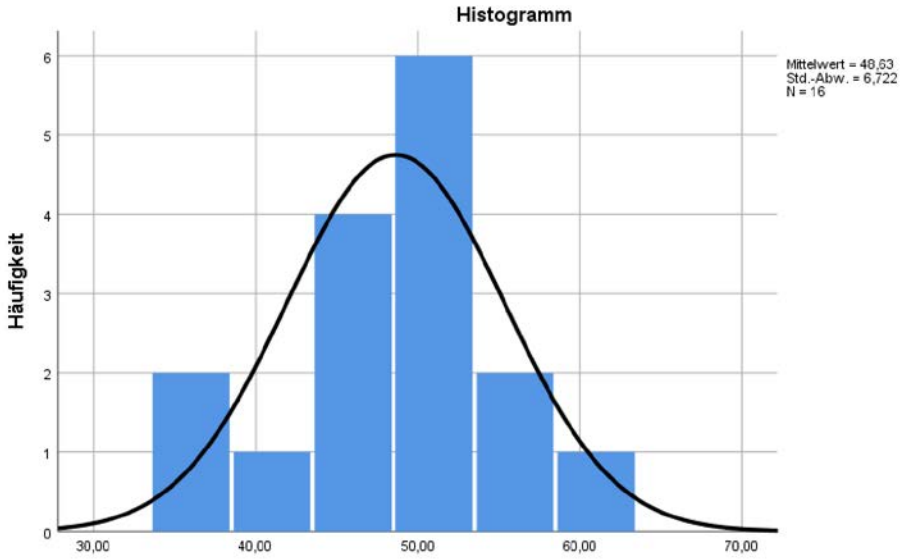

Abbildung 73: Verteilung der Ergebnisse (grafische Gruppe)

Die Analyse der Varianzen beider Stichproben liefert unterschiedliche Varianzen. Zur Überprüfung der Hypothese, ob die grafische Lehrmethode gegenüber der klassischen Lehrmethode etwas bewirkt hat, wird deshalb der Welch-Test auf beide Stichproben angewandt. Dieser liefert folgendes Ergebnis:

Test bei unabhängigen Stichproben

		Levene-Test der Varianzgleichheit		T-Test für die Mittelwertgleichheit						
									99% Konfidenzintervall der Differenz	
		F	Signifikanz	T	df	Sig. (2-seitig)	Mittlere Differenz	Standardfehler der Differenz	Untere	Obere
VAR00001	Varianzen sind gleich	,480	,494	-2,043	30	,050	-5,43750	2,66160	-12,75689	1,88189
	Varianzen sind nicht gleich			-2,043	28,816	,050	-5,43750	2,66160	-12,77717	1,90217

Abbildung 74: Welch-Test Theorie

Bei einem Signifikanzniveau von 5% ist die Hypothese, ob beide Stichproben aus unterschiedlichen Populationen kommen und voneinander verschieden sind zu bestätigen. Somit hat die Unterrichtsmethode auf die Ergebnisse einen Einfluss. (Abbildung 74)

4.4.3 Praxisteil

Der Praxisteil besteht aus drei abgeprüften Teilbereichen. Dem Lesen von Code, dem Entwickeln einer Lösung und dem Programmieren einer Lösung. Diese drei Bereiche bestehen aus insgesamt fünf Beispielen. Für jede Kategorie gibt es zwei Beispiele bis auf die letzte, das „freie" Programmieren. Hier wird ein größeres Beispiel mit mehreren Teilbeispielen programmiert. Die vier Übungen aus den Bereichen „Code lesen" und „Pseu-

docode“ sind mit Multiple Choice Antworten ausgestattet. Der letzte Teil „Programmieren“ wird durch die Funktionsweise der einzelnen Teilbeispiele überprüft.

1. Code lesen
2. Pseudocode
3. Programmieren

4.4.4 Praxisteil zusammenfassend

Die Ergebnisse des Praxisteils sind, zusammenfassend gesagt, über alle praktischen Beispiele hinweg gleichförmig, wie eine Glockenkurve, verteilt. Zu sehen ist diese Notenverteilung in Abbildung 75. Die meisten Noten sind mit Befriedigend vorhanden. Die Anzahl der Gut und Genügend sind ähnlich und auch Sehr Gut und Nicht genügend sind ähnlich verteilt.

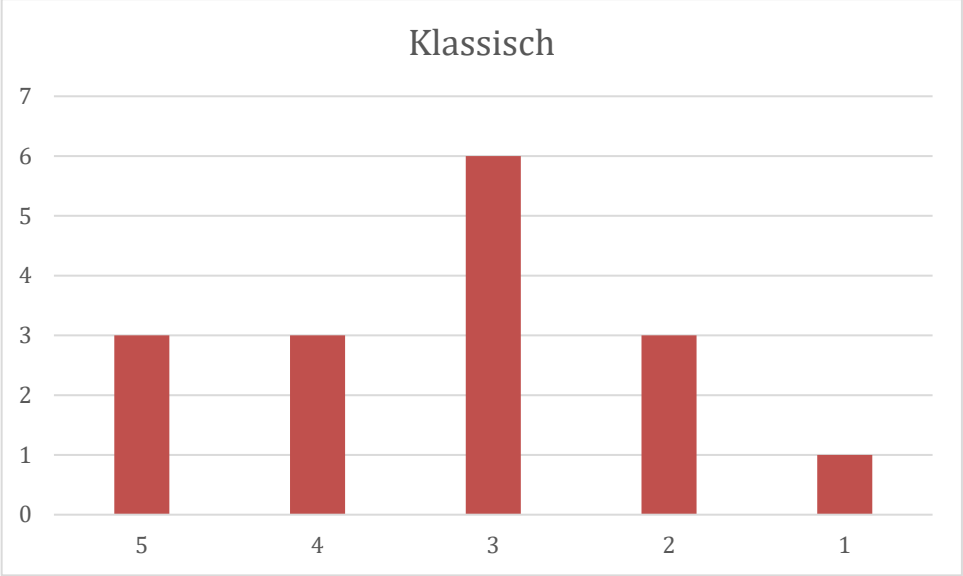

Abbildung 75: Notenverteilung praktisch

Im Anschluss, in Abbildung 76, ist die Notenverteilung vom praktischen Teil der Gruppe „grafische” Programmierung zu finden. Die Noten sind vorwiegend um die Noten Befriedigend und Gut verteilt. Die Sehr Gut sind deutlich weniger als die Befriedigend und Gut. Die Nicht Genügend und Genügend sind gleich verteilt.

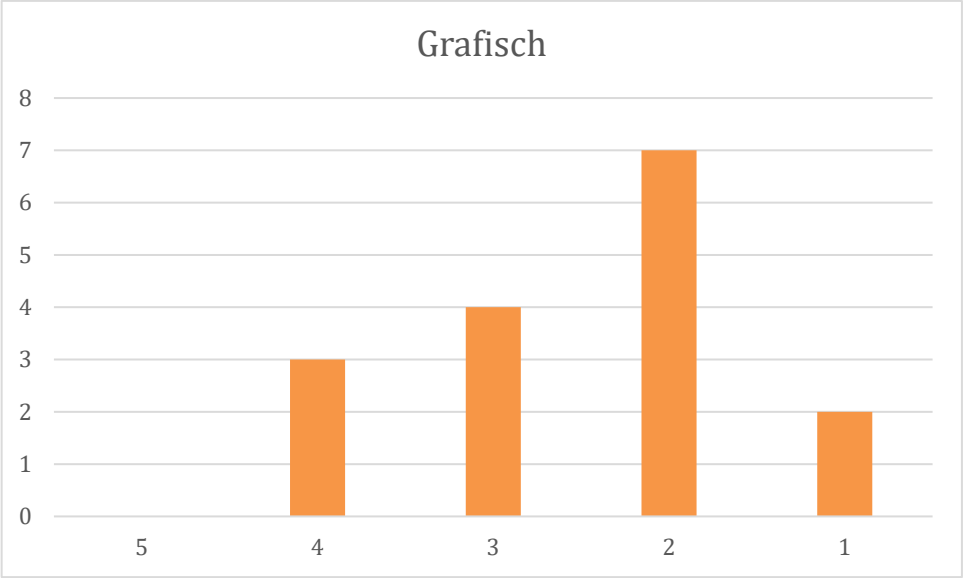

Abbildung 76: Notenverteilung praktisch (grafisch)

Im Vergleich der Notenverteilungen von „klassischer" Programmierung und „grafischer" Programmierung ist zu sehen, dass die Verteilung der Noten bei der „grafischen" Programmierung Richtung Sehr Gut verteilt sind. Die Anzahl der Genügend und Nicht Genügend sind deutlich weniger bei der „grafischen" Programmierung als bei der „Klassischen". Die Anzahl der Gut ist deutlich mehr bei der „grafischen" Programmierung als bei der „klassischen" Programmierung. Eine Tendenz ist zu erkennen und das direkte Verhältnis der beiden Durchschnittsnoten ist bei 1,3, was einer 30% Verbesserung bei der „grafischen" Programmierung gegenüber der „Klassischen" im Vergleich der Noten gleicht.

Das Boxplotdiagramm der Punkteverteilungen der Testung aus Abbildung 77 zeigt beim Praxisteil folgendes Bild:

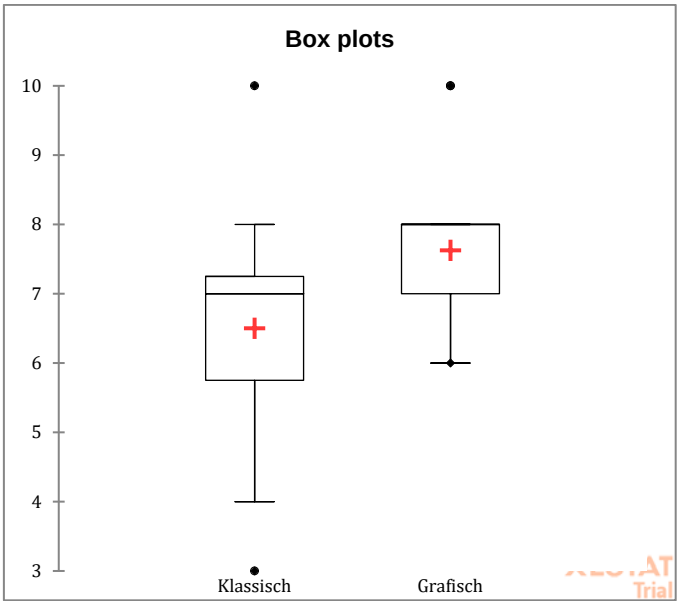

Abbildung 77: Boxplot praktisch

Die Mehrheit der Werte sind beim „klassischen“ Programmieren zwischen 5,5 und 7,5 Punkten und beim „grafischen“ Programmieren zwischen 7 und 8 Punkten. Die Ausreißer beim „klassischen“ Programmieren sind bei 3 für den niedrigsten Wert und 10 für den höchsten Wert. Der Ausreißer beim „grafischen“ Programmieren liegt bei 10 Punkten. Das bedeutet, dass beide Gruppen einmal volle Punkteanzahl bei der Programmierung erreicht haben. Die Verteilung der Werte kann im folgenden Diagramm dargestellt werden:

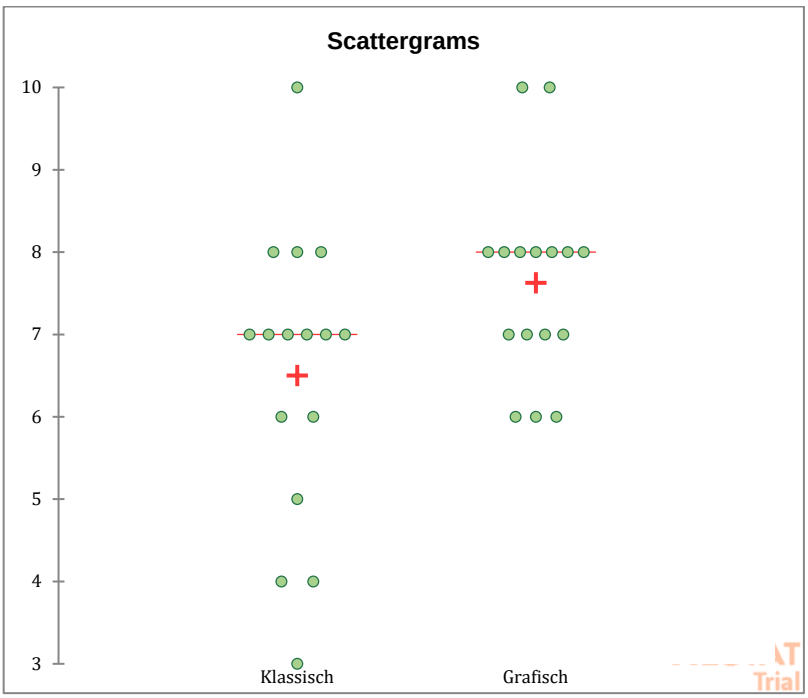

Abbildung 78: Punkteverteilung praktisch

Bei der Verteilung der Werte ist deutlich zu sehen, dass die Mehrheit der erzielten Punkte bei der „grafischen“ Gruppe bei knapp 8 Punkten liegt. Die „klassische“ Gruppe liegt zwischen 6 und 7 Punkten (Abbildung 78).

Die Ausführung des Kolgomorov-Smirnov Test, angewandt auf beide Stichproben, im Praxisteil zeigt (Abbildung 79), dass beide Stichproben gleich verteilt sind. Durch das logarithmische Aussehen der Verteilungen wird mit Hilfe des Kolgomorov-Smirnov Tests auf Normalverteilung geprüft.

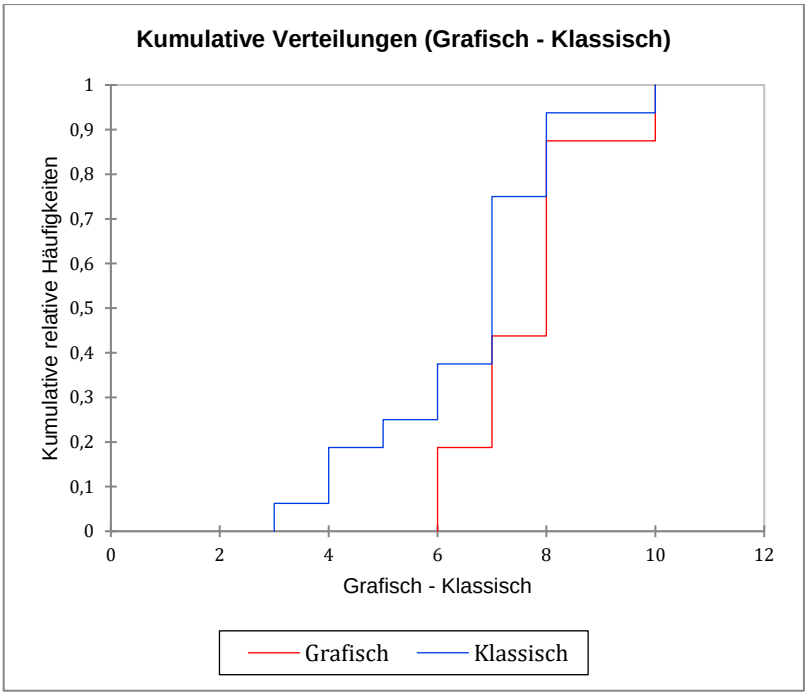

Abbildung 79: Vergleich beider Stichproben - Grafisch - Klassisch – Praxis

Die Normalverteilung wird bestätigt. Auch lassen sich die Verteilungen, wie folgt, grafisch veranschaulichen:

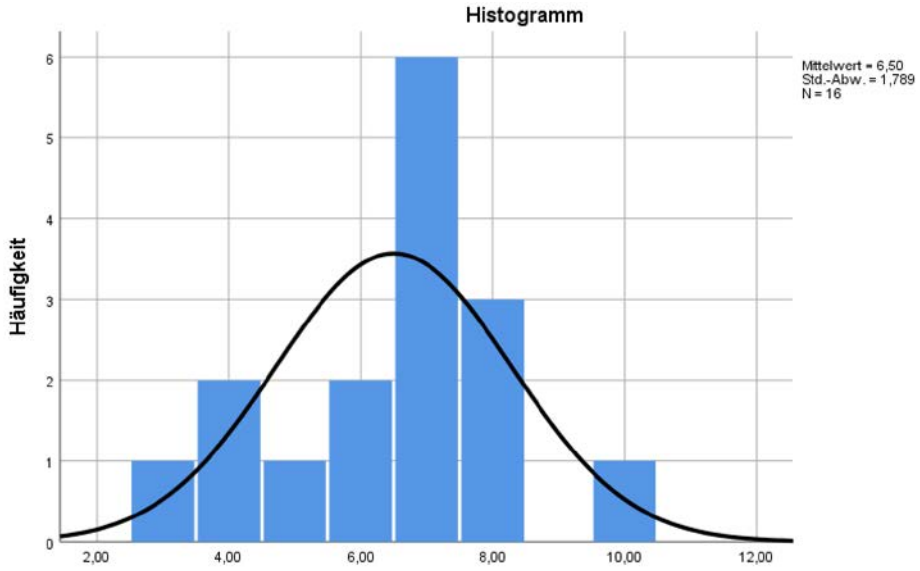

Abbildung 80: Verteilung der Ergebnisse (klassiche Gruppe)

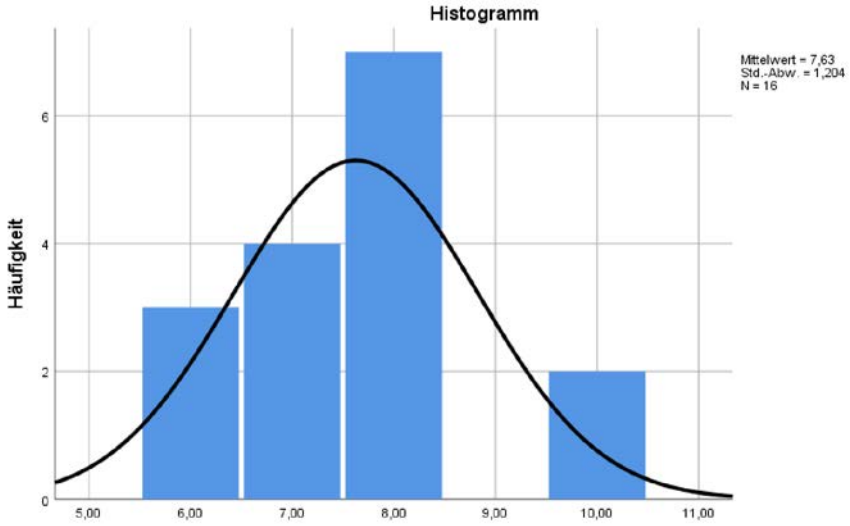

Abbildung 81: Verteilung der Ergebnisse (grafische Gruppe)

Die Analyse der Varianzen zeigt, dass beide Stichproben unterschiedliche Varianzen haben. Aus diesem Grund wird der Welch-Test für beide Stichproben durchgeführt, um die Hypothese, ein Einfluss der Lehrmethode auf die Ergebnisse ist vorhanden, zu bestätigen. Der Welch-Test liefert folgendes Ergebnis:

Test bei unabhängigen Stichproben

		Levene-Test der Varianzgleichheit		T-Test für die Mittelwertgleichheit						
									95% Konfidenzintervall der Differenz	
		F	Signifikanz	T	df	Sig. (2-seitig)	Mittlere Differenz	Standardfehler der Differenz	Untere	Obere
VAR00001	Varianzen sind gleich	1,902	,178	-2,087	30	,046	-1,12500	,53910	-2,22598	-,02402
	Varianzen sind nicht gleich			-2,087	26,278	,047	-1,12500	,53910	-2,23256	-,01744

Abbildung 82: Welch-Test bei praktischem Teil

Das Ergebnis des Welch-Tests, in der zweiten Zeile, Varianzen sind nicht gleich, in obiger Grafik (Abbildung 82), zeigt, dass die Unterrichtsmethode auf die Ergebnisse einen Einfluss hat und somit signifikant (5%iges Signifikanzniveau) ist.

4.4.5 Zusammenfassung der Testung

Zusammenfassend aus den beiden Teilen „klassisch" und „grafisch" sind folgende Testergebnisse insgesamt erzielt worden:

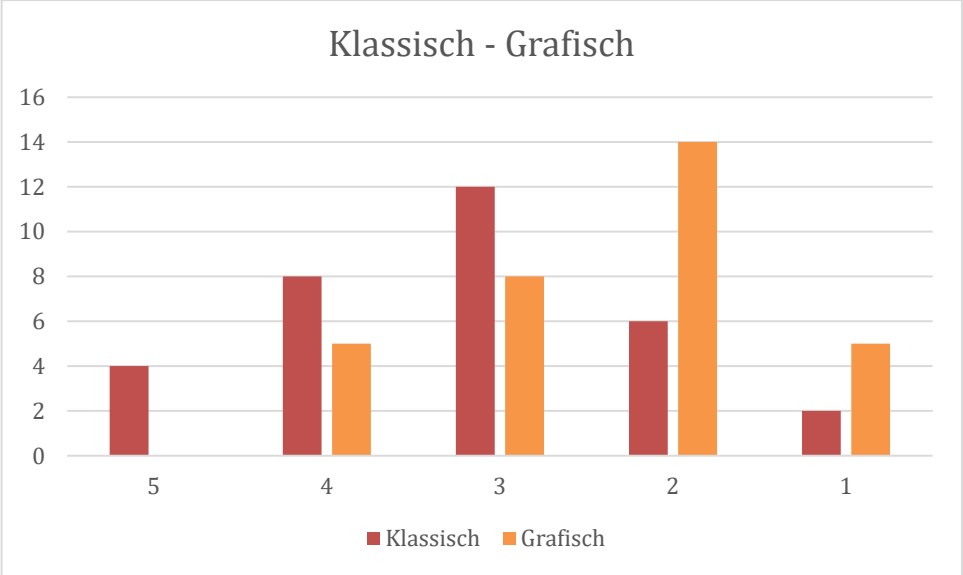

Abbildung 83: zusammenfassende Notenverteilung

Aus dem direkten Vergleich der beiden Notenverteilungen in Abbildung 83 ist zu erkennen, dass die Tendenz beim visuell unterstützten Programmieren in Richtung der Note Gut geht und beim textbasierten Programmieren die Tendenz Richtung Befriedigend geht. Nach den Ergebnissen zu urteilen, sind bessere Programmierkenntnisse in der „grafischen" Gruppe vorhanden als in der „klassischen" Gruppe. Werden die Mittelwerte der Ergebnisse aus den Vortests der Testgruppen mit den Mittelwerten der Ergebnisse aus den abschließenden Testungen verglichen ist eine Verschlechterung der Noten im Schnitt in der „klassischen" Gruppe zu sehen und eine Verbesserung der Noten im Schnitt in der „grafischen" Gruppe zu erkennen. Zu Beginn, bei der Vortestung, ist der Unterschied zwischen den beiden Testgruppen gering, 0,06 zwischen den beiden Mittelwerten. Bei der abschließenden Testung ist der Unerschied zwischen den beiden Testgruppen mit 0,78 zwischen den beiden Mittelwerten zu Gunsten der „grafischen" Gruppe deutlich größer. Eine Studie von mehreren Jahren und oder mehreren Klassen könnte mehr Auskunft über

den Einfluss der visuellen Ausgabe beimProgrammieren geben, denn das Experiment ist natürlich durch die SchülerInnen aus diesem Jahrgang geprägt.

4.5 Erkenntnisse im Unterricht

Der Unterricht in beiden Bereichen, visuell unterstützt und textbasiert, war von sehr unterschiedlicher Motivation geprägt. Zu Beginn waren alle SchülerInnen beider Gruppen aufgeregt und ein hohes Interesse am Programmierenlernen war gegeben. Ebenso war in der ersten Lehreinheit, welche auf verschiedenen Ausgaben beruhte, hohe Motivation vorherrschend.

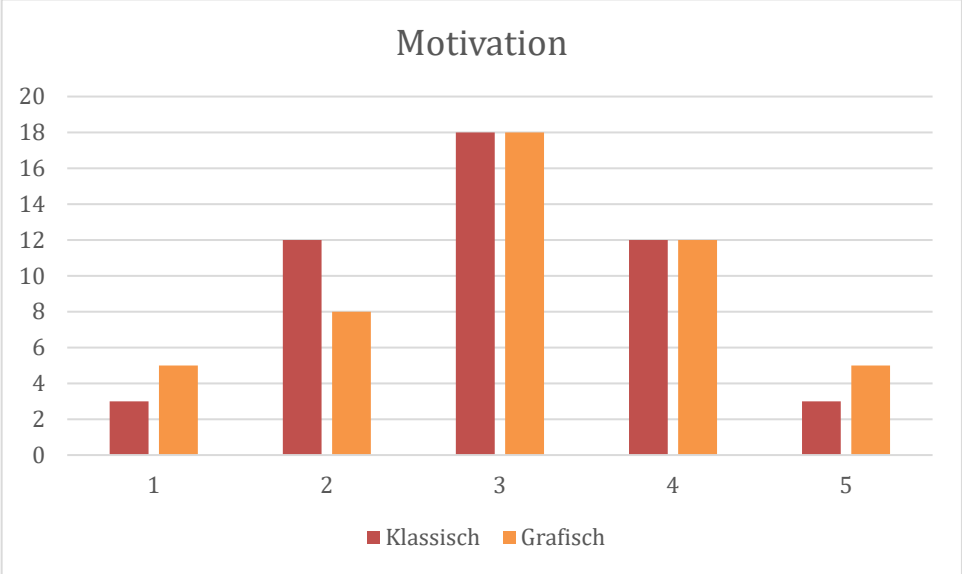

Abbildung 84: Motivationstestung zu Beginn

Die obige Grafik (Abbildung 84) zeigt die Verteilung der Motivation in der „grafischen" und „klassischen" Gruppe. Die Fragen wurden mit fünf Antwortmöglichkeiten ausgestattet von nicht zufrieden bis sehr zufrieden. Die Verteilung der Motivation ist bei beiden Gruppen nahezu ident. Die Mehrheit der Werte liegt in der Mitte zwischen nicht und sehr zufrieden.

Bei der Messung der Zufriedenheit waren die Fragen ebenfalls mit Bewertungen zwischen nicht zufrieden und sehr zufrieden möglich. Der Hauptteil der Werte liegt im Mittel zwischen den beiden extremen, wobei in der „klassischen" Gruppe die Zufriedenheit etwas schlechter gelagert ist (Abbildung 85).

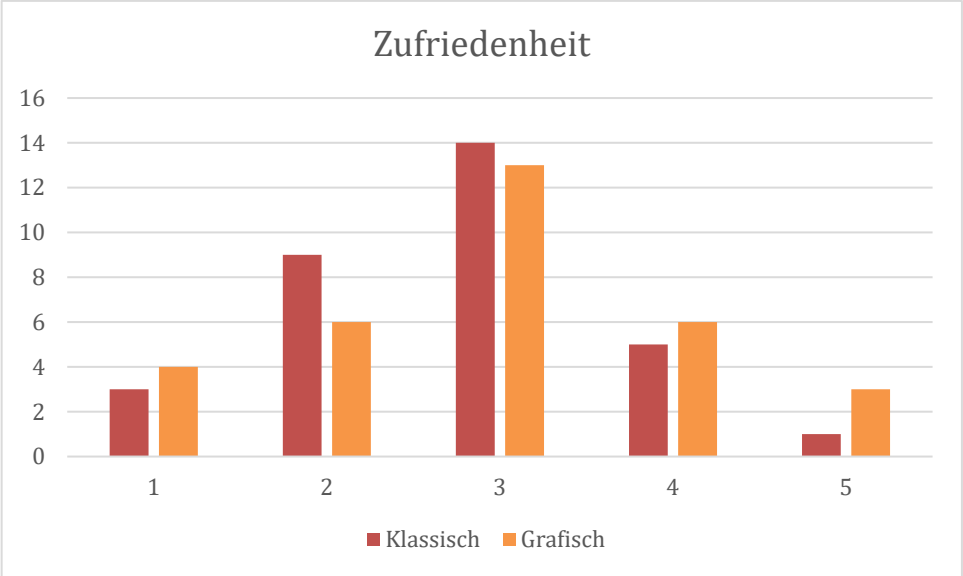

Abbildung 85: Zufriedenheitstestung zu Beginn

Ab der zweiten Lehreinheit sank das Interesse bei der textbasierten Gruppe stark und bei der grafischen Gruppe auch, aber weniger als bei der textbasierten Gruppe. In der dritten Einheit war die Motivation bei beiden Gruppen niedrig. Die grafische Gruppe wirkte etwas aufmerksamer. Die Aufgaben und Beispiele wurden in dieser Einheit etwas komplexer.
In der vierten Einheit stieg das Interesse bei beiden Gruppen an, wobei bei der „grafischen" Gruppe motivierter gearbeitet wurde als bei der „klassischen" Gruppe. Die Motivation in der fünften und sechsten Einheit war bei der visuell basierten Gruppe hoch, da sie Visualisierungen erstellen konnte. Auch bei der textbasierten Gruppe stieg das Interesse in diesen Einheiten wieder.

Die Abschlussprojekte und die Präsentation wurden vorwiegend positiv motiviert erarbeitet und lieferten Ergebnisse. Die letzte Motivationsmessung zeigte folgende Ergebnisse in Abbildung 86:

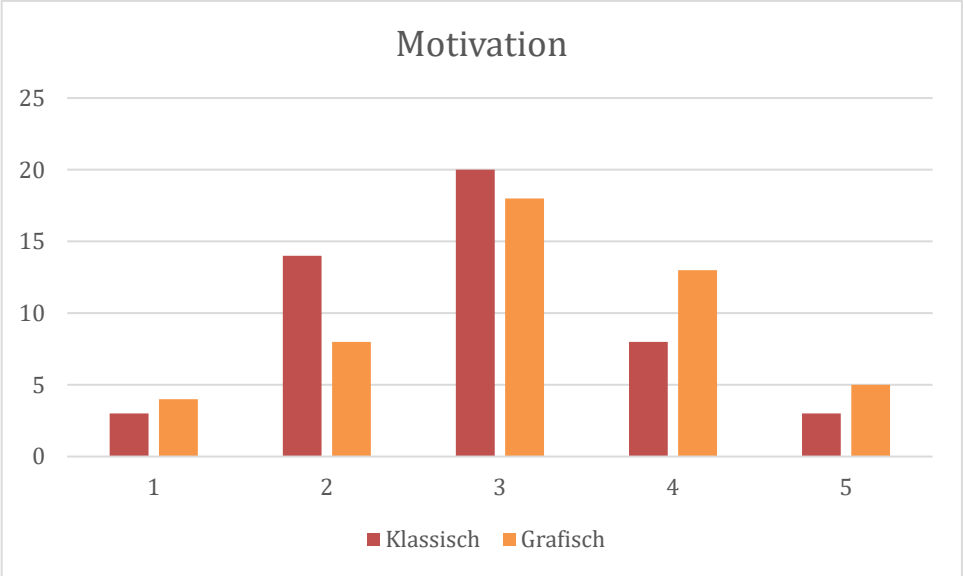

Abbildung 86: Motivationstestung am Ende

Die Motivation scheint aufgrund der häufigen positiven Werte bei der Messung in der „grafischen“ Gruppe höher zu sein. Viele Bewertungen befinden sich zwischen zufrieden und mehr zufrieden. Bei der „klassischen“ Programmierung finden sich die meisten Werte bei zufrieden und weniger zufrieden, was eine verminderte Motivation zeigt.

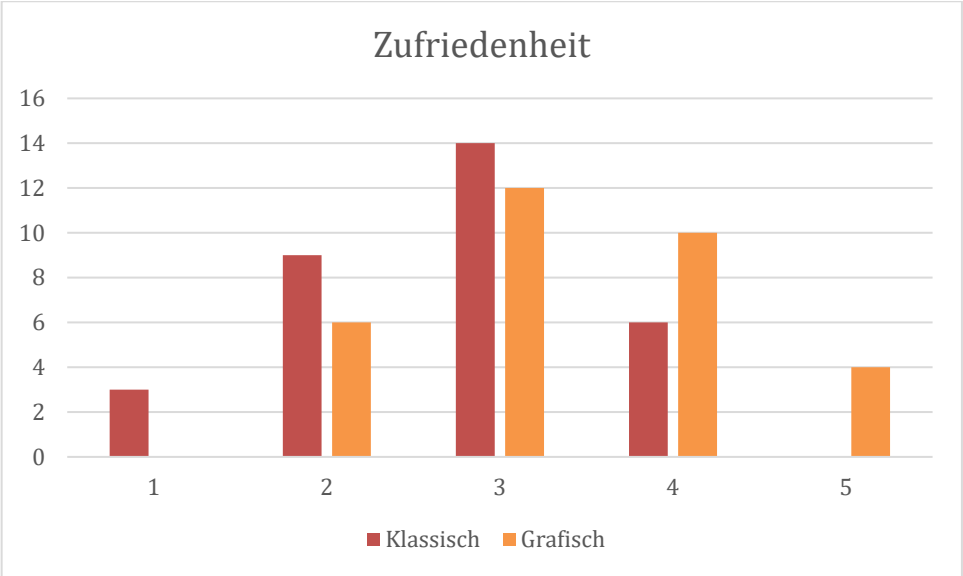

Abbildung 87: Zufriedenheitstestung am Ende

Bei der Zufriedenheit in Abbildung 87 ist zu erkennen, dass die Gruppe „grafische“ Programmierung zufriedener ist als die Gruppe „klassische“ Programmierung. Die Zufriedenheit liegt klar bei der „grafischen“ Gruppe.

4.6 Vergleich zu anderen wissenschaftlichen Arbeiten und Literatur

Ein Vergleich zu anderen wissenschaftlichen Abhandlungen, welche oben erwähnt wurden und ebenfalls eine Effizienzsteigerung von visuell basiertem Programmieren an Universitäten oder an Unterstufen durchgeführt haben, wird hier kurz gezogen.

Im Vergleich zur Ausarbeitung von „Effectiveness of Program Visualization in Learning Java: a Case Study with Jeliot 3" [1] konnte ebenfalls eine Effizienzsteigerung bei der Programmierung mit visueller Ausgabe zur Verständlichkeit bemerkt werden. Basierend auf der obigen Feststellung, dass bei einem Bild beide Gehirnhälften arbeiten und das Lernen angeregt wird und sich Menschen beim Lernen besser Bilder merken können als Text alleine, kann diese These der Effizienzsteigerung bei visueller Ausgabe nach dem Ergebnis bei der Programmierung mit SchülerInnen unterstützt werden.

Wird diese Arbeit mit dem Artikel „What a Novice Wants: Students Using Program Visualization in Distance Programming Course" [15] verglichen, ist ersichtlich, dass das Arbeiten mit grafisch basierten Tools das Lernen von Programmieren unterstützt und den Lernenden ihre Aufgaben deutlich erleichtert. Sowohl diese Arbeit als auch der hier genannte Artikel sind Experimente mit ProgrammieranfängerInnen mit dem Ziel den Einstieg zu verbessern.

Der Artikel "Learning Programming from Scratch" [9] beschäftigt sich mit den positiven Effekten der spielbasierten, visuellen Repräsentation von Code beim Einsatz in der Unterstufe. Die Erkenntnisse in diesem Artikel, dass Spiel und Grafik förderlich für die Lernmotivation und das Verstehen von Code sind, mit der Hilfe der Schaffung von Anreizen, können durch diese Arbeit unterstützt werden.

Das Buch „Programmieren lernen mit Computergrafik" [58] hat zum Ziel den Lernenden das Programmieren durch visuelle Darstellung nahezubringen. Es wird zu Beginn des Buchs ein Überblick über die benötigten Werkzeuge gegeben und Grundbegriffe des Programmierens werden geklärt. Die Kapitel sind aufbauend und in jedem Kapitel werden Visualisierungen durch die Bibliothek Processing und der Programmiersprache Java erstellt, um das Lernen zu unterstützen. Im Vergleich mit dieser Arbeit ist der ähnliche Ansatz sowie der Einsatz der Bibliothek Processing zu erwähnen. Allerdings setzt diese Arbeit auf den webbasierten Ansatz und der Implementierung von Processing in Javascript. Dadurch ist es möglich prozedural, funktional und objektorientiert zu programmieren als

auch HTML beizubringen und somit den Anforderungen des Lehrplans der Informatik zu entsprechen.

5 Konklusion

Das Erlernen des Programmierens mit visueller Unterstützung ist ein interessantes Gebiet, in dem Potential für nachhaltiges, schnelles Lernen steckt. Speziell absolute ProgrammieranfängerInnen profitieren von der grafischen Repräsentation des Programmierten. Einer AnfängerIn mit Basiskenntnissen in EDV die Denkweise einer ProgrammiererIn in kurzer Zeit beizubringen, ist eine Herausforderung. Die Arbeitsweise und das Denken bestehen aus mehreren Faktoren. Diese sind Logik und mathematische Kenntnisse, Grundkenntnisse in EDV und fachliche Qualifikationen wie Kenntnisse über eine Programmiersprache. Beim Lernen selbst ist das wichtigste, das Gelernte schnell und nachhaltig zu memorieren. Unterschiedliche Forschungen haben im Laufe der Zeit gezeigt, dass aktives Lernen mit visuellen Hilfsmittel im Unterricht und die Fähigkeit des Menschen, sich Bilder leicht zu merken, zielführend beim Lernen und Vermitteln von Wissen sind.

In der Informatik gab es an Universitäten bereits Versuche, StudentInnen beim Einstieg ins Programmieren mit grafischen Hilfsmitteln zu unterstützen. Auch an Unterstufen für SchülerInnen im Alter zwischen 10 und 14 Jahren wurde mit Scratch gearbeitet, um ihnen die Arbeitsweise des Programmierens nahezubringen. Sowohl die Experimente an den Universtäten als auch die Versuche in der Unterstufe der Schule haben deutlich gezeigt, dass Programmieren mit visueller Unterstützung effektiver sind als das Programmieren ohne visuelle Unterstützung. Die Lerngeschwindigkeit beim Lerner wird deutlich erhöht und das Verständnis wird besser.

Die Geschwindigkeit beim Erlernen des Programmierens ist an den Oberstufen der Gymnasien an Österreichs Schulen gefordert. Aufgrund der zahlreichen Themengebiete, welche der Schülerin und dem Schüler vermittelt werden, bleibt für das Lernen des Programmierens nicht viel Zeit. Aus diesem Grund ist es in diesem Bereich sehr wichtig, eine effektive und zeitsparende Methode zu entwickeln, welche beim Lernenden nachhaltiges und umfangreiches Wissen erzeugt. Aus diesem Grund wurde hier eine Methode experimentell erprobt, die durch visuelle Ausgabe den Lerner hilft, die Lehrinhalte leichter und einfacher zu verstehen.

Die zwei Testgruppen konnten in den Lehreinheiten die Grundkenntnisse des Programmierens lernen. Durch Theorie, Beispiele und Übungen wurde das Wissen über Program-

mieren in beiden Gruppen erhöht. Der Unterschied zwischen den beiden Gruppen bestand ausschließlich in der textuellen oder visuellen Ausgabe. Eine abschließende Prüfung mit einer Kontrolle der vermittelten Lehrinhalte zeigte die erhofften Ergebnisse: die SchülerInnen mit visueller Ausgabe konnten im Gegensatz zu den SchülerInnen mit textueller Ausgabe eine deutliche Steigerung der Ergebnisse erzielen. Speziell beim praktischen Teil waren die besseren Ergebnisse festzustellen.

Die Methode mit der visuellen Ausgabe ist im Unterricht effizient, da die SchülerInnen sich im Gegensatz zur textuellen Ausgabe an vieles erinnern und viel wiedergeben können. Die SchülerInnen wirken durch die grafischen Inhalte interessierter und motivierter und entwickeln ein besseres Vorstellungsvermögen beim Programmieren. Die Erstellung der angepassten Lehrinhalte ist im Gegensatz zur textuellen Methode mit etwas mehr Aufwand verbunden, allerdings ist dieser Aufwand vernachlässigbar gering. Aufgrund der gewonnen Erkenntnisse und Ergebnisse ist die Methode mit visueller Ausgabe klar der textuell basierten Methode vorzuziehen. Ob diese Methode zu unterrichten zum Standard an Österreichs Schulen wird, ist offen.

Literaturverzeichnis

[1] Čisar, Sanja & Pinter, Robert & Radosav, A & Čisar, Petar. (2011). Effectiveness of Program Visualization in Learning Java: a Case Study with Jeliot 3. International Journal of Computers, Communications and Control. 6. 669-682. 10.15837/ijccc.2011.4.2094.

[2] Wrenn, Jan & Wrenn, Bruce. (2009). Enhancing Learning by Integrating Theory and Practice. International Journal of Teaching and Learning in Higher Education. 21.

[3] Active Learning: Creating Excitement in the Classroom. 1991 ASHE-ERIC Higher Education Reports.

[4] Windschitl, M. (1999). The Challenges of Sustaining a Constructivist Classroom Culture. Phi Delta Kappan. 80. 751-755.

[5] Fayombo, Grace. (2012). Active learning: Creating excitement and enhancing learning in a changing environment of the 21st century. Mediterranean Journal of Social Sciences. 3. 107 - 128. 10.5901/mjss.2012.v3n16p107.

[6] Shaaruddin, Jamila & Mohamad, Maslawati. (2017). Identifying the Effectiveness of Active Learning Strategies and Benefits in Curriculum and Pedagogy Course for Undergraduate TESL Students. Creative Education. 08. 2312-2324. 10.4236/ce.2017.814158.

[7] Garbor Kiss, Teaching Programming in the Higher Education not for Engineering Students, Procedia – Social and Behavioral Sciences, 26 November 2013, Pages 922-927, https://doi.org/10.1016/j.sbspro.2013.10.414

[8] Rugelj, Jože & Zapušek, Matej. (2013). Learning programming with serious games. Transactions on Game Based Learning. 10.4108/trans.gbl.01-06.2013.e6

[9] Mladenovic, Monika & Krpan, Divna & Mladenovic, Sasa. (2017). Learning programming from Scratch, Conference: Conference: International Conference on New Horizons in Education 2017, At Berlin, Germany

[10] Radosevic, Danijel & Orehovački, Tihomir & Lovrencic, Alen. (2009). Verificator: Educational Tool for Learning Programming. Informatics in Education. 8. 261-280.

[11] Jenkins, Tony. (2019). On the difficulty of learning to program.

[12] B. McKeithen, Katherine & Spencer Reitman, Judith & H. Rueter, Henry & Hirtle, Stephen. (1981). Knowledge Organization and Skill Differences in Computer Programmers. Cognitive Psychology. 13. 307-325. 10.1016/0010-0285(81)90012-8.

[13] On the Cruelty of Really Teaching Computing Science, https://en.wikipedia.org/wiki/On_the_Cruelty_of_Really_Teaching_Computer_Science, Abruf 17.02.2019

[14] Boyle,T.; Bradley,C.; Chalk,P.; Jones,R.; Pickard,P;. (2003) *Using blended learning to improve student success rates in learning to program* . Journal of Educational Media, 28 (2-3). pp. 165-178.

[15] O. Kannusmäki, A. Moreno, N. Myller, E. Sutinen. What a novice wants: Students using pro-gram visualization in distance programming course, Proc. of the Third Program VisualizationWorkshop (PVW'04), Warwick, UK, pp. 126-133, 2004

[16] C. D. Hundhausen, S. A. Douglas, J. T. Stasko, A Meta-Study of Algorithm VisualizationEffectiveness, Journal of Visual Languages and Computing, 259-290, 2002.

[17] N. Myller, R. Bednarik, Methodologies for studies of program visualization, Proc. of theMethods, Materials and Tools for Programming Education Conference, 37-42, 2006.

[18] Gustav Pomberger, Heinz Dobler, Algorithmen und Datenstrukturen: eine systematische Einführung in die Programmierung, Pearson Deutschland GmbH, 2008, 576 Seiten, Seite 269ff

[19] Mladenovic, Monika & Krpan, Divna & Mladenovic, Sasa. (2017). Learning programming from Scratch.

[20] Radosevic, Danijel & Orehovački, Tihomir & Lovrencic, Alen. (2009). Verificator: Educational Tool for Learning Programming. Informatics in Education. 8. 261-280.

[21] S. Diehl, Evolution, In Software Visualization: Visualizing the Structure, Behaviour,and Evolution of Software Springer Verlag, pp. 149-160, 2007. [Online]. Available: http://www.springerlink.com/content/m373254212740552/fulltext.pdf

[22] B. A. Price, R. M. Baecker, I. S. Small, An Introduction to Software Visualization, in SoftwareVisualization, J. Stasko, J. Dominque, M. Brown, B. Price (Eds.), London, England MITPress, 4-26, 1998

[23] JELIOT 3, http://cs.joensuu.fi/jeliot/, Abruf am 18.02.2019

[24] Stephan Diehl, Software Visualization, Visualizing the Structure, Behaviour and Evolution of Software, Springer Verlag Berlin Heiderlberg, 2007

[25] Scott R. Tilley and Dennis B. Smith. Coming attractions in program understanding. In Alan W. Brown, editor, Component-Based SoftwareEngineering: Selected Papers from the Software Engineering Institute. Wiley, New York, NY, 1996.

[26] Roger N. Shepard. Recognition memory for words, sentences, and pictures. Journal of Verbal Learning and Behavior, 6:156–163, 1967.

[27] Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A meta-study of algorithm visualization effectiveness. Journal of Visual Languages and Computing, 13(3):259–290, 2002

[28] MatrixPro, http://www.cse.hut.fi/en/research/SVG/MatrixPro/, Abruf am 19.02.2019

[29] Leonardo, http://www.dis.uniroma1.it/~demetres/Leonardo/, Abruf am 19.02.2019

[30] Dale, E. (1969). Audio-Visual Methods in Teaching (3rd ed., p. 108). Holt, Rinehart & Winston, New York: Dryden Press

[31] Močinić, S. N. (2012). Active Teaching Strategies in Higher Education. Metodički-obzori, 15.

[32] Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active Learning Increases Student Performance in Science, Engineering, and Mathematics. Proceedings of the National Academy of Sciences of the United States of America, 111, 8410-8415, https://www.pnas.org/content/111/23/8410

[33] Tedesco-Schneck, M. (2013). Active Learning as a Path to Critical Thinking: Are Compe-tencies a Roadblock? Nurse Education in Practice, 13, 58-60, https://www.sciencedirect.com/science/article/pii/S1471595312001382?via%3Dihub

[34] Dzakiria, H., Don, M. S., & Rahman, H. D. A. (2012). Blended Learning (BL) as Pedagog-ical Alternative to Teach Business Communication Course: Case Study of UUM Execu-tive Diploma Program. Turkish Online Journal of Distance Education, 13, 297-315.

[35] Pani, J.R., Chariker, J.H., Naaz, F. et al. Adv in Health Sci Educ (2014) 19: 507. Learning with Interactive Computer Graphics in the Undergraduate Neuroscience Classroom, https://doi.org/10.1007/s10459-013-9483-3, Springer Netherlands, Print-ISSN 1382-4996

[36] Lehrplan zum Wahlpflichtfach Informatik, https://www.ahs-informatik.com/informatik-wahlpflichtfach/, Abruf am 18.02.2019

[37] Lehrplan Vorschlag 2018: https://www.ahs-informatik.com/app/download/9741860185/Lehrplan-WPF-Informatik-Oktober-21018.pdf?t=15472161373, Abruf am 18.02.2019

[38] W. Cazzola and D. M. Olivares, "Gradually Learning Programming Supported by a Growable Programming Language," *2015 IEEE 39th Annual Computer Software and Applications Conference*, Taichung, 2015, pp. 857-857. doi: 10.1109/COMPSAC.2015.82 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7273711&isnumber=7273573

[39] Y. Hayashi, K. Fukamachi and H. Komatsugawa, "Collaborative Learning in Computer Programming Courses That Adopted the Flipped Classroom," *2015 Interna-*

tional Conference on Learning and Teaching in Computing and Engineering, Taipei, 2015, pp. 209-212.doi: 10.1109/LaTiCE.2015.43, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7126260&isnumber=7126209

[40] G. Bucks and W. Oakes, "Work in progress - impact of graphical programming environments on learning and understanding programming concepts" *2008 38th Annual Frontiers in Education Conference*, Saratoga Springs, NY, 2008, pp. F2A-23-F2A-24. doi: 10.1109/FIE.2008.4720623 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4720623&isnumber=4720249

[41] Fatos Xhafa, Santi Caballé, Ajith Abraham, Thanasis Daradoumis, Angel Alejandro Juan Perez, Computational Intelligence for Technology Enhanced Learning, Springer Science & Business Media, 18.02.2010, 250 Seiten, Seite 103ff

[42] Whitfield, A. K., Blakeway, S., Herterich, G. E., & Beaumont, C. (2007). Programming, disciplines and methods adopted at Liverpool Hope University. Innovation in Teaching and Learning in Information and Computer Sciences, 6(4), 145–168.

[43] Shelley Powers, Einführung in JavaScript, O'Reilly Germany, 2007, 357 Seiten

[44] Michael Schart, Michael Legutke, Lehrkompetenz und Unterrichtsgestaltung, Ernst Klett Sprachen GmbH, 2012, 199 Seiten

[45] Unterrichtsphasen, https://www.guterunterricht.de/unterrichtsphasen, Abruf am 21.02.2019

[46] Einstiegstest für Informatiker, https://www.ausbildungspark.com/einstellungstest/fachinformatiker/, Abruf am 21.02.2019

[47] Eignungstest für Informatiker, https://www.plakos.de/informatik-eignungstest/, Abruf am 21.02.2019

[48] Jörg Bewersdorff, Objektorientierte Programmierung mit JavaScript: Direktstart für Einsteige, Springer-Verlag, 15.02.2018, 379 Seiten, Seite 121ff

[49] Jens Ohlig, Hannes Mehnert, Stefanie Schirmer, Das Curry-Buch: Funktional programmieren lernen mit JavaScript, O'Reilly Media, 22.07.2013, 208 Seiten

[50] Einführung in die Informatik, Heinz Peter Gumm, Manfred Sommer, Walter de Gruyter, 01.12.2010, 912 Seiten, Seite 305ff

[51] Übersicht über die Browserfunktionalitäten, https://kangax.github.io/compat-table/es5,Abruf am 26.02.2019

[52] Säulen der objektorientierten Programmierung, http://www.inztitut.de/blog/glossar/objektorientierte-programmierung/, Abruf am 26.02.2019

[53] Julia Werner, Christian Ebel, Christian Spannagel, Stephan Bayer, Flipped Classroom – Zeit für deinen Unterricht: Praxisbeispiele, Erfahrungen und Handlungsempfehlungen, Verlag Bertelsmann Stiftung, 01.10.2018, 244 Seiten

[54] Processing, https://processing.org/, Abruf am 14.03.2019

[55] Learn Javascript with p5.js, Coding for visual learners, Apress Verlag, Engin Arslan, ISBN: 1484234251, Jahr der Erscheinung 2018, https://doi.org/10.1007/978-1-4842-3426-6

[56] Javascript Quick Syntax Reference, Apress Verlag, Michael Olson, ISBN: 1430264934, Jahr der Erescheinung 2015

[57] ECMAScript, https://en.wikipedia.org/wiki/ECMAScript, Abruf am 22.10.2019

[58] Programmieren lernen mit Computergrafik, Springer Vieweg, Oliver Deussen, Thomas Ningelgen, ISBN 978-3-658-21144-8, Jahr der Erscheinung 2018, https://doi.org/10.1007/978-3-658-21145-5

Anhang

Im Anhang finden sich die benutzten und entwickelten Fragebögen für obige Arbeit wieder.

Einstiegstest zur Analyse der Zielgruppe

Für den Einstiegstest wurden Fragen aus zwei Online-Eignungstest für den Beruf Fachinformatiker und das Aufnahmeverfahren zum Studium der Informatik herangezogen. Diese Fragen wurden aufgrund der erprobten Benutzung ausgewählt und einige finden sich im Test wieder. Beide verwendeten Portale sind auch in den Quellen zu finden: https://www.plakos.de/informatik-eignungstest und https://www.ausbildungspark.com/einstellungstest/fachinformatiker/. Der Aufbau des Tests ist wie folgt in mehrere Kategorien gegliedert und es wurde für jede Frage eine Minute Zeit eingerechnet.

EDV

Was wird in der EDV-Sprache unter „Backup“ verstanden?

- Eingabe rückgängig machen
- Eingabe wiederholen
- Daten zwischenspeichern
- Datensicherung auf Laufwerk oder Festplatte

Wie heißt der Arbeitsspeicher eines Computers?

- Festspeicher
- ROM (Read Only Memory)
- RAM (Random Access Memory)
- Lesespeicher

Wofür stehen die Begriffe analog und digital?

- Für verschiedene Arten von Mikroprozessoren.
- Für Ein- und Ausgabegeräte.
- Für manuelle und automatische Prozesse.
- Für unterschiedliche Signalarten.

Was verstehen Sie unter SQL?

- eine Datenbanksprache
- eine Bildbearbeitungssoftware
- ein Textverarbeitungsprogramm
- eine Tabellenkalkulationssoftware

Was wird in der EDV unter „Partition“ verstanden?

- Die Vorabversion eines Computerprogramms.
- Die einzelnen Musiktracks auf einer CD.
- Die vollständige Löschung der Festplatte.
- Die Aufteilung eines Speichermediums in mehrere Bereiche.

Im Gegensatz zur Berechnung von Integerwerten hat die Berechnung der Gleitkommazahlen

- Keine Nachkommastellen,
- Rundungsfehler,
- einen eingeschränkten Zahlenbereich,
- einen minimalen Speicherbedarf

HTML ist eine Auszeichnungssprache, die

- Am CERN in der Schweiz entwickelt wurde,
- in Cambridge entwickelt wurde,
- am MIT in den USA entwickelt wurde,
- im Silicon Valley entwickelt wurde

Der erste Programmierer war

- Konrad Zuse,
- Ada Lovelace,
- Alan Turing,
- Grace Hopper

In Deutschland gilt als Erfinder des ersten Computers der Welt ein Bauingenieur namens ...

- Konstantin Zuse,
- Konrad Zuse,
- Karsten Zise,
- Karl Zose

Vor der Maus und Tastatur benutzte man zur Programmierung von Computern ...

- Streichhölzer,
- Stampfkarten,

- Lochkarten,
- Mausecodes

Ein Rechner besteht im Prinzip immer aus einer Zentraleinheit (CPU), Speicher und ...

- Netzwerk,
- Ein- und Ausgabeeinheit,
- Tastatur,
- Prozessor

Der Ursprung des Internet liegt im Jahr 1969 in der Entwicklung eines Netzwerks namens ...

- ARGONET,
- DATCOM,
- ARPANET,
- LAMNET

Zahlen und Programme sind gespeichert und dargestellt in ...

- Hexadezimalzahlen,
- Sedezimalform,
- Tertiärform,
- Binärform

Ein Byte ist die Zusammenfassung von

- 6,
- 4,
- 8,
- 10 bit

Die Integerzahl hat genau wie viele Nachkommastellen?

- 0,
- 10,
- 2,
- 1

Wie lautet der Rückgabewert der Funktion Wertumwandlung?

```
function wertumwandlung (EINGABE){
AUSGABE = 0
WENN (EINGABE > 3) AUSGABE = AUSGABE + 5;

WENN (EINGABE > 5) AUSGABE = AUSGABE + 5;

WENN (EINGABE > 7) AUSGABE = AUSGABE + 5;

WENN (EINGABE > 10) AUSGABE = AUSGABE + 5;


return AUSGABE;
}
```

Mathematik

Bei der Betriebsratswahl der Max Mayer Einzelhandelsgesellschaft sind von 100 Beschäftigten 85 Prozent wahlberechtigt. Wie viele Beschäftigte dürfen wählen?

- 60 Beschäftigte
- 70 Beschäftigte
- 75 Beschäftigte
- 85 Beschäftigte

Von 85 wahlberechtigten Beschäftigten haben 68 bei der Betriebsratswahl gewählt. Wie viel Prozent der wahlberechtigten Beschäftigten haben gewählt?

- 50 %
- 60 %
- 70 %
- 80 %

Bei der letzten Betriebswahl lag die Wahlbeteiligung bei 90 % und es haben 81 Beschäftigte gewählt. Wie viele wahlberechtigte Beschäftigte hatte die Firma damals?

- 80 wahlberechtigte Beschäftigte
- 82 wahlberechtigte Beschäftigte
- 88 wahlberechtigte Beschäftigte
- 90 wahlberechtigte Beschäftigte

Am Betriebsausflug haben vier Mitarbeiter nicht teilgenommen. Das sind fünf Prozent der Belegschaft. Wie viele Beschäftigte hat der Betrieb?

- 70 Beschäftigte

- 75 Beschäftigte
- 80 Beschäftigte
- 85 Beschäftigte

Herr Mayer hat für eine Betriebsversammlung einen Raum inklusive Bewirtung angemietet. Da Herr Mayer Stammkunde ist, erhält er das Angebot 15 % rabattiert für 3.400 €. Wie hoch wäre der reguläre Preis gewesen?

- 4.000 €
- 4.500 €
- 5.000 €
- 5.500 €

6 - 3 + 2 × 3 = ?

- 15
- -15
- 9
- -5

6 - (3 + 2) × 3 = ?

- 3
- 15
- -9
- -15

(6 - (3 + 2)) × 3 = ?

- 3
- 15
- -9
- -15

-3^3 = ?

- 3
- 9
- -27
- -6

-3^2 = ?

- -9
- 9
- 6
- -6

Das Hexadezimalzahlsystem ist ein Zahlensystem welches aus...

- 2,
- 4,
- 8,
- 16 Zahlen besteht

Die binäre Zahl 111 ist in Dezimal ausgedrückt die Zahl

- 8
- 6
- 7
- 3

2 hoch 10 sind genau ...

- 1024,
- 2048,
- 256,
- 512

Die binäre Zahl 01101101 ist in Dezimal ausgedrückt die Zahl ...

- 64,
- 164,
- 109,
- 128

Die kleinste Maßeinheit zur Speicherung oder Übertragung von Daten heißt ...

- 1kb,
- bit,
- byte,
- 1MG

Die Dezimalzahl 9 ist binär die Zahl ...

- 0011
- 0101
- 1001
- 0101

Logik

9 / 4 8 / 5 7 / 6 6 / 7 ?

- 3 / 7
- 4 / 8
- 5 / 8

- 9 / 4

6 18 36 5 15 30 4 ?

- 12
- 20
- 3
- 34

9 27 24 72 69 207 ?

- 144
- 132
- 138
- 204

144 36 72 18 36 ?

- 42
- 56
- 54
- 9

42 38 40 38 38 38 ?

- 30
- 38
- 42
- 36

Die ODER Verknüpfung aus W(0) und W(1) ergibt

- 4
- 2
- 0
- 1

Wenn Klaus sich nicht ärgert, weiß er nichts vom erdbeerfreien Erdbeereis.

- Klaus liebt Erdbeereis nicht.
- Klaus ärgert sich nicht.
- Klaus ärgert sich.

Welche Schlussfolgerung ist logisch richtig, wenn die folgende Behauptung zugrunde gelegt wird? „Peter arbeitet gerade oder liest ein Buch. Peter liest gerne Geschichtsbücher, aber heute liest er kein Buch."

- Peter arbeitet nicht.
- Peter arbeitet.

- Peter liest ein Buch.
- Peter liest ein Buch, wenn er arbeitet.

Welche Schlussfolgerung ist logisch richtig, wenn die folgende Behauptung zugrunde gelegt wird? „Wenn Claudia Klaus liebt, dann wird sie ihm vorschlagen, gemeinsam eine Wohnung zu mieten. Claudia schlägt Klaus vor, gemeinsam eine Wohnung zu mieten."

- Klaus liebt Claudia nicht.
- Klaus liebt Claudia.
- Claudia liebt Klaus.
- Claudia liebt Klaus nicht.
- Keine Antwort ist richtig.

Welche Schlussfolgerung ist logisch richtig, wenn die folgende Behauptung zugrunde gelegt wird? „Alle Bilder sind Gemälde. Gemälde sind schön. Einige Gemälde sind Kunst. Also …".

- ist nicht jedes Bild ein Gemälde.
- ist jedes Bild ein Gemälde und Kunst.
- ist alles, was schön ist, ein Bild.
- ist ein Bild ein Gemälde und schön.
- Keine Antwort ist richtig.

Welche Schlussfolgerung ist logisch richtig, wenn die folgende Behauptung zugrunde gelegt wird? „Peter möchte heute entweder fernsehen oder Alex besuchen. Peter besucht Alex. Also …".

- sieht Peter fern.
- sieht Peter mit Alex fern.
- sieht Peter kein fern.
- sieht Alex fern.
- Keine Antwort ist richtig.

Theoriefragen der Testung

Die Theoriefragen der Testung basieren auf den vermittelten Inhalten in den Lehreinheiten. Durch die Testung wird das erlernte Wissen in der Programmierung abgefragt. Es handelt sich bei den Theoriefragen nicht um ausschließliches Wissen über die Programmiersprache Javascript. Es werden alle Kenntnisse im Bereich der Programmierung, welche in den Unterrichtsstunden gewonnen werden konnten, abgefragt. Es handelt sich um Multiple Choice Fragen, für welche pro Frage eine Minute Zeit kalkuliert wurde. Die Fragen sind wie folgt gegliedert.

Teil1

Der Teil 1 beschäftigt sich mit den einführenden Begriffen.

- Was ist alles Teil der Javascript Programmierung?
- Welche Operatoren in Javascript gibt es?
- Wie ist eine Javascript Variablenzuweisung aufgebaut?
- Was sind reserved words?
- Was sind Datentypen von Javascript?
- Was kann mit dem typeof Operator bestimmt werden?
- Was ergibt der Vergleich "2" === 2?
- Welche logischen Vergleichsoperatoren gibt es?
- Was sind gültige Variablennamen?
- Welche Anweisungen sind Konvertierungen von Werten?

Teil2

Der Teil 2 prüft das Wissen über die Kontrollstrukturen und Vergleichsoperatoren.

- Was sind Kontrollstrukturen?
- Was sind Codeblöcke?
- Was macht folgende while-Schleife? Var i = 0; While (i < 100) { console.log('RUNNING') }; console.log('END');
- Was ist ein Inkrement oder Dekrement?
- Wozu dient ein if-Statement?
- Was ist call-by-value?
- Wozu brauche ich break in einem switch-Statement?
- Wozu brauche ich das continue in einer Schleife?
- Wozu brauche ich ein logisches UND?
- Was ist dynamische Typisierung?

Teil3

Der Teil 3 beschäftigt sich mit dem Thema Funktionen und Scopes.

- Wie wird eine Funktion definiert?
- Was ist ein return value?
- Was sind globale Variablen?
- Wozu dienen Parameter?
- Welche Scopes gibt es?
- Welche Funktionsnamen sind gültig?
- Wozu braucht man Code Guidelines?
- Was ist die Aufgabe von Funktionen?
- Was passiert, wenn lokale und globale Variablen denselben Namen besitzen?
- Was sind rekursive Funktionen?

Teil4

Der Teil 4 beschäftigt sich mit dem Thema Objekte.

- Was sind Objekte?
- Was bedeutet das Attribut private?
- Was bedeutet Instanzieren?
- Was sind Klassen?
- Was ist die Punkt-Notation?
- Welche Grundpfeiler in der objektorientierten Programmierung gibt es?
- Wozu dienen getter und setter Methoden?
- Was ist ein Konstruktor?
- Wozu dient prototype?
- Was kann mit dem this Keyword gesteuert werden?

Teil5

Im 5. Teil wird das Wissen über Arrays abgefragt.

- Was ist ein Array?
- Was kennzeichnet einen Index?
- Was macht die Methode valueOf?
- Was ist ein assoziativer Array?
- Was ist kennzeichnet einen Stack?
- Was kann mit shift erreicht werden?
- Wie lösche ich einen Wert aus einem Array?
- Was ist ein Dictionary?

- Wie lässt sich ein Array erzeugen?
- Was passiert, wenn auf das erste Element eines leeren Arrays zugegriffen wird?

Teil6

Im 6. Teil wird das Wissen über Algorithmen überprüft.

- Was ist ein Algorithmus?
- Was ist ein Suchalgorithmus?
- Welche Sortieralgorithmen gibt es?
- Was ist funktionale Programmierung?
- Wozu brauche ich die Funktion map?
- Was ist ECMAScript?
- Was ist Pseudocode?
- Was ist der Unterschied zwischen funktionaler und prozeduraler Programmierung?
- Was sind Typisierungen?
- Was ist Programmieren?

Fragen zur Messung der Motivation und Zufriedenheit

Ein Bereich der Arbeit beschäftigt sich mit der Motivation und Zufriedenheit. Nach einer Recherche zur Messung dieser Komponenten wurde ein Fragebogen mit folgenden Fragen erstellt und die Fragen mit einer Antwortskala von nicht motiviert bis voll motiviert in fünf Stufen und nicht zufrieden bis voll zufrieden in fünf Stufen ausgestattet. An unterschiedlichen Zeitpunkten wurden diese Fragen gestellt, um einen Verlauf dieser Komponenten zu beobachten und die Auswertung ist in obiger Ausarbeitung zu finden. Die Fragen lauteten:

- Wie sehr erfreut dich das Programmieren?
- Wie beurteilst du deinen Lernfortschritt?
- Wie hoch schätzt du dein Wissen ein?
- Wie zufrieden bist du mit dem Unterricht?
- Was hältst du von den Lehrmaterialien?