

# Ein kombiniertes Rahmenmodell zum Programmierenlernen

## Entwicklung, Erprobung und Evaluation einer Lehrveranstaltung für das Programmierenlernen unter Benutzung von Jupyter Notebook

Robert Ringel<sup>1</sup> und Hermann Körndle<sup>2</sup>

**Abstract:** Der Beitrag zeigt das Konzept eines Rahmenmodells zum Programmierenlernen, welches auf Grundlage anerkannter lernpsychologischer Methodiken entwickelt wurde. Im Kern steht die Bearbeitung von Aufgabenfolgen gemäß der 4C/ID-Methodik. Sie wird von Prozessen der Reflexion und Erkenntnissicherung begleitet. Nach diesem Rahmenmodell wurde ein Grundlagenkurs zur Programmierausbildung an der HTW Dresden durchgeführt. Ausgewählte quantitative Ergebnisse und qualitative Aussagen von Studierenden zeigen Erfolge und Möglichkeiten der Weiterentwicklung des vorgeschlagenen Rahmenmodells.

**Keywords:** Programmierenlernen, Aufgabenfolgen, 4C/ID, Rahmenmodell, Methodik

## 1 Einleitung

Das Programmierenlernen ist seit vielen Jahren fester Bestandteil der Ingenieurs- und Informatikausbildung an Hochschulen. Das Lernziel besteht in der Befähigung, grundlegende ingenieurtechnische Probleme mit Hilfe von Computerprogrammen zu lösen. Seit den 1980er Jahren erfolgt eine wachsende Erforschung der Lehr-Lern-Prozesse in der Informatik, die sich insbesondere auf den Bereich des Programmierenlernens fokussiert. Robins [Ro19] hat für den Zeitraum 1980-2018 über 200 Quellen aus diesem Bereich analysiert. Im Handbuchkapitel mit dem Titel „Novice programmers and introductory programming“ stellt er die Charakteristika des Programmierens, die Probleme der Lernenden, Aussagen zu Lernergebnissen sowie generelle Hinweise zum Lehren und Lernen des Programmierens anhand der wissenschaftlichen Befundlage zusammen. Bedeutsam ist dabei seine Feststellung, dass es unter den Lehrenden kein einheitliches Verständnis zum elementaren methodischen Vorgehen in Grundlagenkursen des Programmierens gibt. Robins geht noch weiter, in dem er feststellt, dass nicht einmal ein Konsens über die wesentlichen Fachinhalte und deren begründete Sequenzierung im Lernprozess besteht.

Dieser Beitrag skizziert die Konzeption eines Rahmenmodells zum Programmierenlernen, welches auf der Grundlage anerkannter lernpsychologischer Modelle entwickelt wurde. Der Beitrag zeigt die daraus resultierende Implementierung einer Lehrveranstaltung zum Erlernen der Python-Programmierung. Praktische Erfahrungen, die Darstellung von

---

1 HTW Dresden, Fakultät Informatik/Mathematik, PF 120 701, 01008 Dresden, Deutschland, robert.ringel@htw-dresden.de

2 TU Dresden, Fakultät Psychologie, 01062 Dresden, Deutschland, hermann.koerndle@tu-dresden.de

Prüfungsleistungen sowie Aussagen von Studierenden belegen das Potential des Rahmenmodells.

## **2 Entwurf eines lernpsychologisch fundierten Rahmenmodells zum Programmierenlernen an Hochschulen**

Aebli [Ae98] beschreibt in seinem Standardwerk „Zwölf Grundformen des Lehrens“ den vollständigen Lernzyklus. Er besteht aus den Phasen: Aufbauen, Durcharbeiten, Üben und Wiederholen sowie Anwenden. Darauf aufbauend schlagen Wespi, Luthiger, Wilhelm [WLW15] ein allgemeines Vorgehensmodell zur Gestaltung kompetenzorientierter Aufgabensets vor, welches aufgabenbasiertes Lernen in eben diesen vollständigen Lernzyklen ermöglicht.

Van Merriënboer und Kirschner [MK18] entwickeln mit Four-Component Instructional Design (4C/ID) eine aufgabenbasierte Methodik, die eine konkrete Umsetzung zur Gestaltung von Aufgabensets nach [WLW15] darstellt. Das Ziel von 4C/ID besteht darin, komplexes Lernen zu ermöglichen und einen umfassenden, zusammenhängenden Aufbau von Wissen, Handlungsfähigkeit und Problemlösekompetenz zu gewährleisten. Dabei kommen ganzheitliche Aufgabenstellungen mit fachpraktischem Hintergrund zum Einsatz. Im Kern der Methodik stehen Folgen von Lernaufgaben, die mit unterstützender Information und Anleitungen zu Handlungsabläufen unteretzt sind. Die Aufgabenanordnung erfolgt so, dass eine nachlassende aufgabeninhärente Unterstützung den Anforderungsgrad systematisch steigert. Gleichzeitig sorgt die inhaltliche Variabilität der Aufgabenschwerpunkte dafür, die Herausbildung und Festigung von Konzeptwissen und Problemlösestrategien zu ermöglichen. Für das Programmierenlernen schlägt van Merriënboer [Va90] vor, zunächst Aufgaben zum Programmverstehen und zum Nachprogrammieren bekannter Lösungen zu bearbeiten. Erst gegen Ende des Lernzyklus sollen Programme allein auf Grundlage der Aufgabenstellung vollständig neu geschrieben werden.

Task-Based Learning (TBL) nach Willis [Wi04] ist eine aufgabengestützte Methodik aus dem Bereich des Fremdsprachenlernens. TBL benutzt jedoch nach dem Zyklus der Aufgabenbearbeitung einen dedizierten Lernschritt zur Erkenntnissicherung und Fokussierung auf sprachliche Korrektheit. Im letzten Lernschritt erfolgt nochmals reale sprachliche Anwendung. Damit hebt TBL das Defizit einer fehlenden Konsolidierung oder Erkenntnissicherung von 4C/ID auf.

Die Kombination beider Methodiken führt zu einem kombinierten Rahmenmodell gemäß Abbildung 1. Dabei bildet die TBL-Methodik mit den Phasen Pre-Task, Task, Post-Task und der abschließenden Complex-Task den Rahmen, in welchen die Aufgabenfolgen gemäß der 4C/ID-Methodik eingebettet sind.

Vor Beginn der Arbeit an den Aufgabenfolgen sollten die Lernenden durch ein Video oder durch einen fachlichen Vortrag grundlegendes Wissen zum Lernbereich erworben haben. In

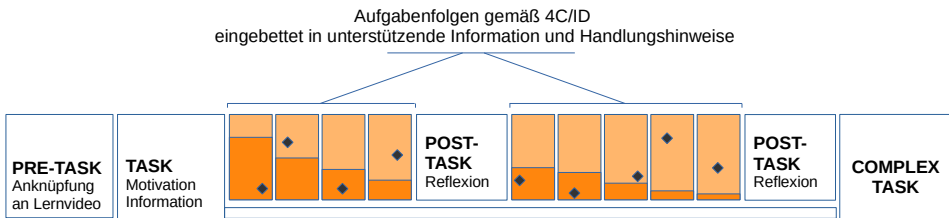


Abb. 1: Schematische Darstellung des kombinierten Rahmenmodells

einer Pre-Task-Phase zu Beginn der Aufgabenfolge wird der Anschluss daran hergestellt, indem wesentliche Inhalte des Videos rekapituliert werden. Dazu führt die Lehrkraft im Auditorium ein kurzes Live-Coding mit den neuen Programmiererelementen durch. In der Task-Phase wird die fachliche Motivation für den Themenbereich aufgebaut, indem die Lernenden überlegen, welche Möglichkeiten für die fachliche Anwendung aus den neuen Programmiermöglichkeiten erwachsen. Entsprechende Erkenntnisse werden schriftlich im Arbeitsmaterial festgehalten. Der Informationsteil der Task-Phase schafft den Zugang zu weiteren Informationsquellen des jeweiligen Fachthemas. Dies können Verweise auf Buchkapitel oder entsprechende Webseiten sein. Diese unterstützende Information steht während der folgenden Aufgabenbearbeitung zur Verfügung. Danach beginnt die Bearbeitung der Aufgabenfolge mit variablen inhaltlichen Schwerpunkten und einer rückläufigen aufgabeninhärenten Unterstützung in Form von vorgegebenem Quelltext oder integrierten Code-Kommentaren. Wichtig ist nach einer Anzahl von 3-5 Lernaufgaben im Bereich Post-Task eine Reflexion zu den Aufgabenergebnissen und ggf. eine explizite schriftliche Erkenntnissicherung vorzunehmen. Am Ende der Aufgabenfolge eines Themenbereichs steht eine Komplexaufgabe. Damit wenden die Studierenden das Gelernte in einer umfangreichen, realen Fachaufgabe problemlösend an. Dieser Zyklus einer Aufgabenfolge ist für jeden Lernbereich des Programmierlehrgangs zu durchlaufen.

Wichtig für den Erfolg der Methodik ist eine große Anzahl Lernaufgaben mit ausreichender Variabilität der Aufgabenstellung. Dem gegenüber steht der klassische Lehrbetrieb an Hochschulen. Eine Woche besteht in der Regel aus einem Vorlesungsanteil und einem Übungsanteil. Die eigene Lehrerfahrung in der Informatik zeigt, dass eine praktische Übungszeit von 90 Minuten pro Woche für die Mehrzahl der Studierenden nicht ausreicht, um zu den Vorlesungsinhalten gefestigte praktische Fähigkeiten aufzubauen. Daher empfiehlt es sich, das vorgeschlagene Rahmenmodell im Lehrbetrieb so zu realisieren, dass pro Woche mindestens zweimal 90 Minuten aktives Lernen an den Aufgabenfolgen stattfinden kann. Dazu werden die Vorlesungsinhalte in Themenvideos verfügbar gemacht. Die Videos im zeitlichen Umfang von 15-20 Minuten müssen die Lernenden vor Beginn einer Aufgabenfolge angeschaut haben, um die fachliche Einführung in das Themengebiet zu erhalten. Die Arbeit an der Aufgabenfolge beginnt dann mit einer kurzen Erinnerung an Kerninhalte des Videos. Durch den Wegfall der Vorlesung zugunsten einer weiteren Übungsstunde haben die Lernenden innerhalb einer Woche an drei Tagen die Möglichkeit, sich mit

dem Programmierenlernen zu beschäftigen. Dabei wirkt regelmäßige Programmierarbeit, idealerweise im Rhythmus von 2-3 Tagen, dem natürlichen Vergessenprozess systematisch entgegen.

### 3 Gestaltung der Aufgabenfolgen

Eine Grundlage für die Entwicklung von Aufgabenfolgen einer Lehrveranstaltung bilden die Lernziele. Van Merriënboer und Kirschner schlagen vor, die Lernziele in einer Hierarchie anzuordnen [MK18, S. 14]. Diese Hierarchie ist der Ausgangspunkt für die nachfolgende Sequenzierung der Ziele. Daher ist die Leserichtung des Baumes auf der Blattebene von links nach rechts definiert. Lernziele, deren Erarbeitung parallel integriert erfolgt, sind durch Doppelpfeile zu kennzeichnen. Die Abbildungen 2 und 3 zeigen die Lernzielhierarchie für den Kurs zu den Grundlagen der Pythonprogrammierung. Durch die Doppelpfeile an den Knoten „Beherrschung der Programmiersprache“ und „Entwicklung von Problemlösungen“ wird gezeigt, dass das Problemlösen integrierter Bestandteil der Beherrschung der Programmiersprache ist. Das Qualifikationsziel dieses Kurses besteht in der problemlösenden Anwendung der Programmiersprache Python.

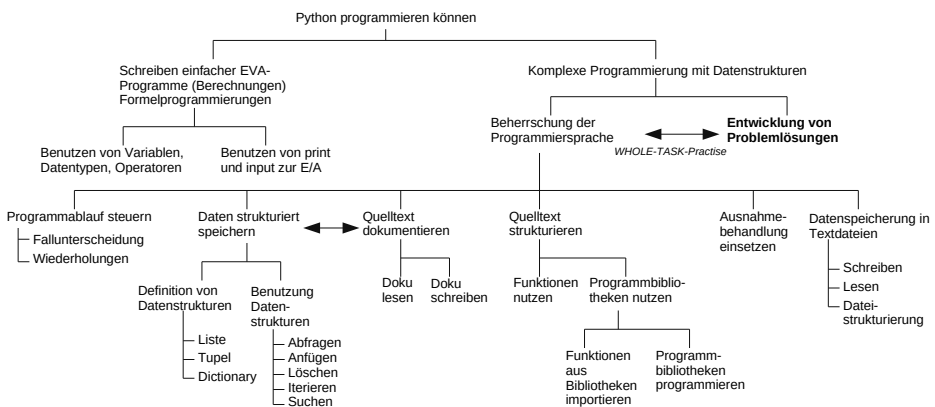


Abb. 2: Beherrschung der Programmiersprache als Teil 1 der Lernzielhierarchie des Grundlagenkurses zum Python-Programmieren

Aus der hierarchischen Darstellung wird nun der Lehrveranstaltungsplan für die jeweilige Zahl der Semesterwochenstunden abgeleitet und beispielsweise in Tabellenform festgehalten. In der Tabelle sollten die Lernziele in den Dimensionen Wissen, Handlungsfähigkeit, zu lösende Probleme untersetzt werden, um so, in Anlehnung an die Kompetenzdefinition der Arbeitsgruppe „Computing Curricula 2020“ [CC21, S. 126], eine zielgerichtete Kompetenzentwicklung zu erreichen. Diese Informationen sind zugleich wesentliche Inhaltsangaben zur Gestaltung der Lernaufgaben. Außerdem lässt sich anhand des Plans ableiten, welche fachlichen Themen als Lehrvorträge oder Lehrvideos zu welchen Zeitpunkten erforderlich sind, um die notwendigen Grundkenntnisse für die Bearbeitung der Programmieraufgaben

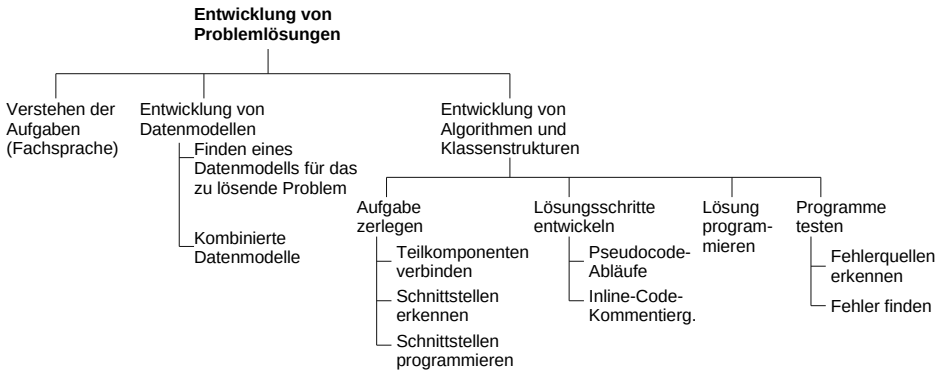


Abb. 3: Entwicklung von Problemlösungen als Teil 2 der Lernzielhierarchie des Grundlagenkurses zum Python-Programmieren

bereitzustellen. Zudem soll die Tabelle zeigen, welche zeitlichen Umfänge für die jeweiligen Aufgabenfolgen zur Verfügung stehen. Damit ist der formale Rahmen dargestellt, der durch die Folgen von Lernaufgaben zu füllen ist.

Im nächsten Arbeitsschritt wird für jeden Lernbereich gemäß der Tabelle ein Pool von Lernaufgaben entwickelt. Dabei besteht das Ziel darin, solche Aufgaben zu gestalten, die das Fachwissen festigen, Konzeptverständnis fördern und notwendige Handlungssicherheit entstehen lassen [KP23]. Dies bildet die Grundlage, um mit Hilfe komplexer Aufgabenstellungen fachliche Probleme im jeweiligen Themenbereich zu bearbeiten. **Dabei sind die Handlungen der Lernenden im Zuge der Aufgabenbearbeitung das entscheidende Element für die Gestaltung lernwirksamer Aufgabenstellungen.**

Tabelle 1 zeigt einen Aufgabenpool aus dem Abschnitt Datenstrukturen des Grundlagenkurses zur Python-Programmierung. Das Lernziel dieses Kursabschnitts besteht im problemlösenden Anwenden der Datenstrukturen Liste, Tupel und Dictionary. Die Tabelle macht deutlich, wie die Lernziele, Lernhandlungen und die Aufgabentypen der einzelnen Aufgaben aufeinander abgestimmt sind. Insgesamt sind für den Themenbereich acht Aufgaben ansteigender Schwierigkeit zur Bearbeitung verfügbar, bevor am Ende eine Komplexaufgabe bearbeitet wird. Im Ergebnis der Aufgabenlösung dieser Komplexaufgabe wird die erreichte Kompetenz der Lernenden im Kursabschnitt sichtbar. Die Darstellung und Ausarbeitung des zugrundeliegenden Kompetenzmodells nach [CC21, S.126] geht über den Rahmen dieses Beitrags hinaus. Sie ist jedoch ein wertvolles Instrument zur Beurteilung der erreichten Lernziele.

Für die Implementierung der Aufgabenfolgen gemäß dem kombinierten Rahmenmodell eignen sich auch digitale Lehr-Lern-Umgebungen. Dies sollte insbesondere bei Grundlagenkursen zum Programmierenlernen bedacht werden [SB23]. Die Aufgabeninhalte und die Aufgabenumfänge der Programmieraufgaben von Grundlagenkursen machen es nicht

Aufgabe	Lernziel	Lernhandlungen	Aufgabentyp	Anmerkung
A	Listen ersetzen Skalarvariablen	lesen, gedanklich verändern, Notizen aufschreiben	Worked- Out- Example	Partner- gespräch
B	Programmierung Liste	Teile implementieren, testen	Comple- tion	-
C	Inhomogene Listen	Code lesen, vergleichen, ausführen, Erkenntnis notieren	Worked- Out- Example	Partner- gespräch
D	Problemlösung mit Listen	Datenstruktur skizzieren, Lösungsansatz finden, Codestruktur entwerfen, Code implementieren, testen	Conven- tional Task	Muster- lösung, Reflexion: Problem- to-Code
E	Zusammengesetz- ter Key für Dictionary; Dictionary speichert Liste	lesen, Kurzkomentare schreiben, Programm- funktion beschreiben	Worked- Out- Example	Partner- gespräch
F	Programmierung Dictionary	lesen, gedanklich verändern, implementieren, testen	Comple- tion	Muster- lösung
G	Programmierung Set	Begriff erarbeiten, Begriff transferieren, Code implementieren	Comple- tion	Lösung für alle darstellen
H	Grundoperationen aller Datenstrukturen	lesen, Details erkennen, markieren, vergleichen, klassifizieren	Worked- Out- Example	Lösung für alle darstellen
Complex Task	Problemlösen mit Datenstrukturen	Datenstruktur skizzieren, Problemlösung entwickeln, Codestruktur entwerfen, implementieren, testen	Conven- tional Task	Gestuftes Hilfe- system, Lösung für alle darstellen

Tab. 1: Übersicht zum Aufgabenpool des Themenbereichs der Datenstrukturen

erforderlich, professionelle Entwicklungswerkzeuge einzusetzen. Mit Jupyter Notebook steht eine einfache Programmierumgebung zur Verfügung, welche nach [Re20], [Se20] und [Za19] erfolgreich für die Lehre einsetzbar ist. Ein Vorteil von Jupyter Notebook besteht darin, dass Aufgabentexte, ergänzende Informationen und grafische Darstellungen gemeinsam in einem Medium mit dem ausführbaren Programmquelltext integriert sind. Dabei können alle Arten von Informationen durch die Lernenden genauso bearbeitet werden wie der Programmcode. Damit ist Jupyter Notebook ein wirklich interaktives digitales Lernmedium. Die grundlegenden Arbeitsmaterialien werden durch die Lehrkraft gestaltet und im Kurs publiziert. Sie bilden dabei unmittelbar das Konzept des kombinierten Rahmenmodells zum Programmierenlernen gemäß Abbildung 1 ab. Im Zuge der Aufgabenbearbeitung tragen die Lernenden ihre Arbeitsergebnisse in diese Materialien ein. Insbesondere die Möglichkeit zur individuellen Reflexion und zur schriftlichen Erkenntnissicherung machen Jupyter Notebook zu einem wertvollen Lernwerkzeug, da es für die Lernenden hilfreich ist, neben der reinen Aufgabenlösung auch Erkenntnisse festzuhalten. Die Art der Erkenntnisgewinnung kann als eine Reflexion von Lösungsvarianten im Partnergespräch stattfinden. Das Beschreiben der eigenen Lösungen bzw. das Verstehen einer alternativen Lösung stellen dabei wertvolle Lernhandlungen dar. Diese Handlungen sind wichtige Ergänzungen, die über den unmittelbaren Problemlösungsprozess hinausgehen.

## 4 Ergebnisse und praktische Erfahrungen

Der folgende Abschnitt zeigt Ergebnisse und praktische Erfahrungen bei Durchführung der Lehrveranstaltung „Grundlagen der Python-Programmierung“ im Lehrbetrieb der Hochschule. Ausgehend von den organisatorischen Rahmenbedingungen des Lehrbetriebs werden die Ergebnisse quantitativer und qualitativer Evaluierungen dargestellt. Das Kapitel endet mit einer kritischen Reflexion zu den Erfahrungen im Lehrbetrieb.

### 4.1 Erfahrungen aus dem praktischen Lehrbetrieb der Hochschule

Der Python-Lehrgang wurde nach dem oben dargestellten methodischen Konzept des kombinierten Rahmenmodells (s. Abb. 1) gestaltet und mit Hilfe von Jupyter-Notebook implementiert. Er wurde in zwei Jahrgängen des Bachelor-Studiengangs Wirtschaftsingenieurwesen im 1. Semester als Pflichtmodul für jeweils 70-80 Studierende im Wochenumfang von zwei Doppelstunden an der Hochschule für Technik und Wirtschaft Dresden durchgeführt. Als Voraussetzung mussten alle Studentinnen und Studenten einen eigenen Notebook-Computer mit einer Jupyter-Notebook-Installation verfügbar haben. Anders als zunächst angenommen war dies kein Problem. Innerhalb einer Woche waren alle Studierenden damit ausgestattet und arbeitsfähig. Auf dieser Grundlage konnten die Lehrveranstaltungen gemäß dem Stundenplan im Umfang von zweimal 90 Minuten pro Woche sowohl im Hörsaal als auch in Seminarraum jeweils als praktische Programmierübungen durchgeführt werden. Für den Hörsaal bedeutet dies, eine solche Sitzordnung einzurichten, dass die Lehrkräfte bei

individuellen Fragen und Problemen, die einzelnen Studierenden erreichen können. Auch dies war alsbald gängige Praxis im Hörsaal.

In beiden Kursdurchgängen zeigte sich nach einer kurzen Eingewöhnungsphase eine aktive, von gegenseitigem Vertrauen geprägte Lernatmosphäre. Das Bestreben, in diesem neuartigen, auf die aktive Tätigkeit der Lernenden gerichteten Format, praktisch programmieren zu lernen, war bei der überwiegenden Mehrzahl der Studierenden deutlich erkennbar. Die Anregung, sich bei Fragen und Problemen gegenseitig zu helfen und Aufgabenlösungen mit dem Sitznachbarn zu besprechen wurde gern angenommen. Diese sozialen Interaktionen wurden fester Bestandteil der Arbeitskultur des Kurses.

Gemäß Moduldefinition der Hochschule muss diese Lehrveranstaltung mit einer schriftlichen Prüfung abschließen. Da es zum Zeitpunkt des Kurses nicht möglich war, für 80 Personen eine isolierte, digitale Prüfungsumgebung mit Jupyter Notebook aufzusetzen, musste die Prüfung mit Papier und Stift geschrieben werden. Dies war ein erheblicher Nachteil, da die Lernenden durch Jupyter Notebook natürlich ein echtes interaktives Programmieren inklusive Syntax-Hervorhebung und Testmöglichkeit für den Quelltext gewohnt waren. Um diesen Nachteil etwas auszugleichen, wurden in beiden Jahrgängen im Laufe des Semesters drei Lernstandserhebungen in schriftlicher Form durchgeführt. Hier hatten die Studierenden die Möglichkeit, sich in Bezug auf Inhalt, Anforderungsniveau und Art der Aufgabenstellungen für die Abschlussprüfung vorzubereiten. Zudem gaben die Lernstandserhebungen sowohl den Studierenden als auch der Lehrkraft eine zuverlässige Rückmeldung zur Erreichung der Lernziele. Die Lernstandserhebungen hatten den Umfang einer Doppelstunde. Die nachfolgende Lehrveranstaltung wurde dazu genutzt, dass die Lehrkraft die Lösung jeder Testaufgabe inklusive einer Anleitung zur Punktergabe vorstellte. Damit war es möglich, dass alle Lernenden ihre eigene Lösung möglichst objektiv nach einem vordefinierten Punktschema bewerten konnten. Die Punktergebnisse der Aufgaben wurden über ein digitales Formular erfasst und ergaben so ein Gesamtbild des Lernstandes im Kurs, welches für die Lehrkraft und die Lernenden einsehbar war.

Diese Art der Lernstandserhebung als Feedback-Instrument für die Lernenden und Lehrenden und gleichzeitig als methodisches Mittel zur systematischen Klausurvorbereitung, wurde von den Lernenden als wertvoll und zweckmäßig empfunden. Neben dem Test des eigenen Könnens verwiesen die Studierenden dabei auf die Lernwirkung der nachfolgenden intensiven Lösungsbesprechung und der damit verbundenen Transparenz bei der Punktergabe.

Die Prüfung selbst diagnostizierte Lernergebnisse in den unterschiedlichen Teilkompetenzen des Programmierens. Konzeptionelles Verständnis wurde anhand von Begriffsdefinition, Begriffsnetzwerken und der Zuordnung entsprechender Python-Anweisungen erfasst. Die Fähigkeiten im Problemlösen, in Handlungsfähigkeit und Transferleistungen waren in verschiedenen Teilaspekten auf die einzelnen Aufgaben verteilt. Insgesamt waren in 120 Minuten vier fachliche Aufgabenstellungen mit insgesamt zehn Teilaufgaben anhand vorgegebener Python-Quelltexte zu bearbeiten. Hierbei war überwiegend vorgegebener Pro-



grammquelltext zu dokumentieren und zu vervollständigen. Zudem waren Beispielskizzen der verwendeten Datenstrukturen anzufertigen. Der Anteil vollständig neu zu schreibenden Quelltexts belief sich auf wenige Funktionen im Umfang von 8-12 Codezeilen.

## 4.2 Ergebnisse der Evaluierungen in zwei Studienjahrgängen

Abbildung 4 zeigt die Boxplots der Prüfungsergebnisse beider Lehrveranstaltungsdurchgänge. Der Punktwert des Klausurergebnisses im Intervall 0-1 repräsentiert den Anteil erreichter Punkte bezogen auf die mögliche Maximalpunktzahl der Prüfung.

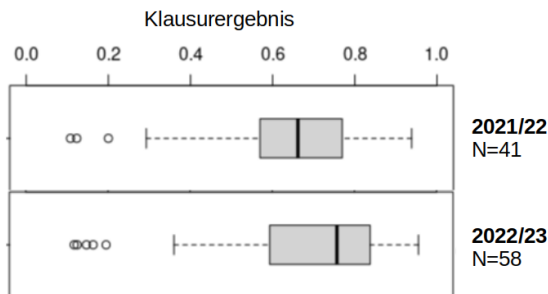


Abb. 4: Boxplots der Prüfungsergebnisse für zwei Lehrveranstaltungsdurchgänge

Für den Durchgang 2022/23 zeigt sich insgesamt ein besseres Prüfungsergebnis. Dies wird durch den höheren Medianwert von 0.75 und durch die höher liegenden Grenzen der Q1- und Q3-Quartile im Boxplot sichtbar.

Damit eine Möglichkeit eröffnet wird, das Lernergebnis des Kurses in Bezug zum Vorwissen der Studierenden darzustellen, wurde in der ersten Lehrveranstaltung auf freiwilliger Basis der Stand der Vorkenntnisse erfasst. Hierfür wurde den Kursteilnehmern eine Liste mit 30 Fachbegriffen des Programmierens vorgelegt. In dieser Liste waren jene Begriffe zu markieren, die so gut bekannt sind, dass die Überzeugung besteht, den jeweiligen Fachbegriff einer anderen Person erklären zu können.

Abbildung 5 zeigt zwei Quadranten-Plots, in denen das begriffliche Vorwissen in Bezug zum Klausurergebnis dargestellt wird. Von besonderem Interesse sind die beiden unteren Quadranten rechts (A) und mittig (B). In ihnen sind die Datenpunkte für jene Studierenden aufgetragen, die geringe bis mittlere begriffliche Vorkenntnisse hatten und die den Programmierkurs mit einem mittleren bis guten Ergebnis abschließen konnten. Es handelt sich also um jenen Teil der Kursgruppe, der ein positives Lernergebnis erzielt hat. Im unteren rechten Quadranten (A) befindet sich der Anteil von Personen mit einem guten Lernergebnis. Für das WS 2021/22 sind es 15% und für das Folgejahr 43%. In den unteren mittleren Quadranten (B) fallen die Indikationen einer mittleren Leistung. Es sind dies 44% für das WS2021/22 und 32% für das Folgejahr.

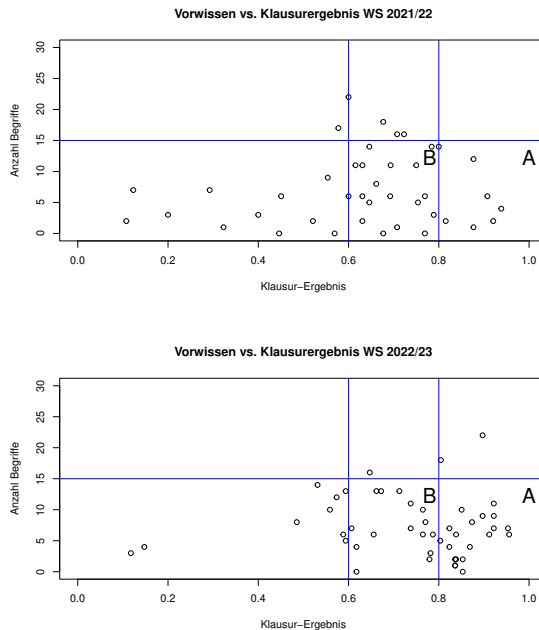


Abb. 5: Quadrantencharts der Prüfungsergebnisse in Bezug zum Vorwissen

Ergänzend zu den quantitativen Befunden anhand der Prüfungsleistungen wurden mit zufällig ausgewählten Kursteilnehmern Interviews als Leitfadengespräche in Anlehnung an [DBP16, S.372-373] durchgeführt. Neben einer Vielzahl individueller Details Aussagen finden sich wiederkehrende Äußerungen, deren Grundaussage in vielen Gesprächen getroffen wird. Einige Beispiele sollen an dieser Stelle aufgeführt werden.

Die Idee, durch den Einsatz von Lernvideos eine Möglichkeit zu schaffen, pro Woche zweimal aktiv zu programmieren, wird durchweg positiv aufgenommen. Ebenso wird der Einsatz von Jupyter Notebook als interaktives Lernwerkzeug für die Lehrveranstaltung und für das individuelle Arbeiten zu Hause sehr begrüßt. Im Wesentlichen wird Jupyter Notebook für das aktive Programmieren genutzt. Deutlich seltener machen die Studierenden ohne Aufforderung darin individuelle Eintragungen außerhalb der Lernaufgaben.

Die Lernstandserhebungen werden ebenfalls durchgängig begrüßt. Sie stellen nach Meinung der befragten Personen eine gute Möglichkeit der Überprüfung des eigenen Lernstands dar. In diesem Zusammenhang werden auch die Darstellung der Aufgabenlösung und die Transparenz zur Punktvergabe als hilfreich benannt.

Der Inhalt und die Gestaltung der Lernvideos werden überwiegend positiv bewertet. Dabei wird wiederholt angegeben, dass man sie im Gegensatz zu einer Vorlesung mehrfach anschauen kann, dass man sie unterbrechen, den Inhalt selbst am Computer nachvollziehen

kann und danach den Film fortsetzen kann. Einzelne Personen geben an, dass einige Videos zu lang oder zu „trocken“ sind.

Methodische Elemente, die mehrfach positiv als wichtiger Bestandteil des Lehrkonzepts benannt werden, sind: das Live-Coding zum Einstieg in die Aufgabenfolge, die Vielzahl der Übungsaufgaben und das gestufte Hilfesystem in den Jupyter Notebooks sowie die Partnerarbeit. Das Anforderungsniveau der Lehrveranstaltung wird von den meisten Befragten als angemessen bis anspruchsvoll eingeschätzt.

### 4.3 Kritische Reflexion der Erfahrungen und Ergebnisse

Kritische Erfahrungen aus den beiden Durchgängen der Lehrveranstaltung betreffen die folgenden Bereiche: Es war zu beobachten, dass ein bestimmter Anteil der Lernenden wiederholt in die Lehrveranstaltung kam, ohne das Lernvideo angeschaut zu haben. Dabei zeigte sich, dass es für diese Personen deutlich schwieriger war, die Aufgabenfolgen zu bearbeiten, da natürlich das einführende Programmierbeispiel allein das Lernvideo nicht kompensieren kann.

Die reine Bearbeitung der Aufgabenfolgen reicht für viele Studierende nicht aus, um nachhaltige Erkenntnisse aufzubauen. Daher ist der Praxis einer regelmäßigen Ergebnissprechung und Lösungsdiskussion eine große Bedeutung beizumessen. Unterbleibt dies, so wurde wiederholt bemerkt, dass die Lernenden bei der Bearbeitung der Aufgabe 4 nicht mehr wissen, welche Erkenntnis aus Aufgabe 2 entstanden ist. Daher wurden im zweiten Durchgang der Lehrveranstaltung in den Jupyter Notebooks extra farblich hervorgehobene Bereiche zur Erkenntnissicherung ergänzt. An diesen Stellen wurde während der Praktika grundlegendes Konzeptwissen in Form von einfachen Merksätzen formuliert.

Im Rahmen des Problemlöseprozesses gehen die Studierenden häufig viel zu schnell zum Programmieren über. Die Aufforderung zum Anfertigen von Skizzen zu den Datenstrukturen als Basis der Problemlösung steht explizit im Aufgabentext. Dennoch wird sie immer wieder einfach übergangen. Zukünftig wäre es interessant, solche Aufgaben zu konstruieren, bei denen lediglich die Schritte der Problemanalyse und der Lösungskonstruktion verlangt werden, nicht aber die Formulierung des Programmcodes.

Generell fällt den Studienanfängern das Denken in Konzepten und Strategien schwer. Studierende mit einem Vorstudium oder mit einem beruflichen Hintergrund haben augenscheinlich weniger Schwierigkeiten bei diesen Anforderungen.

Obwohl die Lehrveranstaltung insgesamt überwiegend positive Lernergebnisse geliefert hat und die Mehrzahl der Studierenden am Ende über wesentliche Grundkenntnisse im Programmieren verfügt, bleibt ein erhebliches Risiko für den nachhaltigen Erfolg. Damit die erworbenen Kenntnisse und Fähigkeiten nicht verblassen, ist es unbedingt erforderlich, in den nachfolgenden Semestern in anderen Lehrveranstaltungen die Programmierfähigkeit systematisch einzufordern und zu praktizieren.

Positiv bleibt zu vermerken, dass wesentliche Ideen aus dem Design dieses Kurses, aus den Ideen der Aufgabengestaltung und zur Nutzung von Jupyter Notebook durch Kollegen aufgegriffen wurden, wodurch sie positiven Einfluss auf die Gestaltung weiterer Lehrveranstaltungen nehmen.

## 5 Zusammenfassung

Das kombinierte Rahmenmodell zum Programmierenlernen adaptiert lernpsychologisch fundierte Methodiken des aufgabenbasierten Lernens. Seine Lernwirksamkeit wurde in Grundlagenkursen der Python-Programmierung erfolgreich evaluiert. Dabei hat sich gezeigt, dass aufgabenbasiertes Lernen ohne Vorlesung erfolgreich durchführbar ist und dass diese Methodik von den Lernenden gern angenommen wird.

Jupyter Notebook ist bestens geeignet, um interaktive Lernmaterialien für Grundlagenkurse des Programmierenlernens zu erstellen. Es unterstützt neben den unmittelbaren Handlungen des Programmierens auch Lernhandlungen der individuellen Reflexion und Erkenntnissicherung. Diese wichtigen Lernhandlungen sind als fester Bestandteil in die Methodik des kombinierten Rahmenmodells integriert und in den Aufgabenfolgen verankert.

## Literatur

- [Ae98] Aebli, H.: Zwölf Grundformen des Lehrens eine allgemeine Didaktik auf psychologischer Grundlage; Medien und Inhalte didaktischer Kommunikation; der Lernzyklus. Klett-Cotta, Stuttgart, 1998.
- [CC21] CC-2020 Task Force: Computing Curricula 2020: Paradigms for Global Computing Education. Association for Computing Machinery, New York, NY, USA, 2021.
- [DBP16] Döring, N.; Bortz, J.; Pöschl-Günther, S.: Forschungsmethoden und Evaluation in den Sozial- und Humanwissenschaften. Springer, Berlin, 2016.
- [KP23] Körndle, H.; Proske, A.: Arbeitsplatznahe Lernaufgaben: Ihre Modellierung, Konstruktion und Einsatz in digitalen Lehr-Lernszenarien beruflichen Lernens. bwp@ Profil 8: Netzwerke – Strukturen von Wissen, Akteuren und Prozessen in der beruflichen Bildung. Digitale Festschrift für Bärbel Fürstenau zum 60. Geburtstag/, 2023, URL: [https://www.bwpat.de/profil-8\\_fuerstenau/koerndle-proske](https://www.bwpat.de/profil-8_fuerstenau/koerndle-proske).
- [MK18] Merriënboer, J. J. G. v.; Kirschner, P. A.: Ten steps to complex learning a systematic approach to four-component instructional design. Routledge, Taylor & Francis Group, New York, 2018.
- [Re20] Reades, J.: Teaching on Jupyter. REGION 7/, S. 21–34, 2020.

- [Ro19] Robins, A. V.: Novice Programmers and Introductory Programming. In: The Cambridge Handbook of Computing Education Research. Cambridge Handbooks in Psychology, Cambridge University Press, S. 327–376, 2019.
- [SB23] Schmolitzky, A.; Burau, H.: OPPSEE - Eine Online-Plattform zum Programmieren Üben, SEUH 2023, 2023.
- [Se20] Seddighi, M.; Allanson, D.; Rothwell, G.; Takrouiri, K.: Study on the use of a combination of IPython Notebook and an industry-standard package in educating a CFD course. Computer Applications in Engineering Education 28/, 2020.
- [Va90] Van Merriënboer, J. J. G.: Strategies for programming instruction in high school: Program completion vs. Program generation. Journal of Educational Computing Research 6/, S. 265–285, 1990.
- [Wi04] Willis, J.: A framework for task-based learning. Longman, Harlow, 2004.
- [WLW15] Wespi, C.; Luthiger, H.; Wilhelm, M.: Mit Aufgabensets Kompetenzaufbau und Kompetenzförderung ermöglichen. Haushalt in Bildung & Forschung 4/4, S. 31–46, 2015.
- [Za19] Zastre, M.: Jupyter Notebook in CS1: An Experience Report. In: Proceedings of the Western Canadian Conference on Computing Education. WCCCE '19, Association for Computing Machinery, Calgary, AB, Canada, 2019.