Ebel, M. · Barkela, V. Reuter, T. Stiel-Dammer, S. Leuchter, M. Weber, A.

HANDBUCH ZUM BILDUNGSANGEBOT

PROGRAMMIERE. DAMIT DIR EIN LICHT AUFGEHT – COMPUTATIONAL THINKING MIT CALLIOPE MINI

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

McSAiK FoSach

Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

bitte zitieren als:

Ebel, M., Barkela, V., Reuter, T., Stiel-Dämmer, S., Leuchter, M., & Weber, A. M. (2023). *Handbuch zum Bildungsangebot "Programmiere, damit dir ein Licht aufgeht - Computational Thinking mit Calliope mini".* Landau: Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau.

Inhalt

1. Über diese Handreichung

Diese Handreichung umfasst Informationen zu einem informatischen Bildungsangebot zum Thema *Computational Thinking*, das in Grundschulen in den Klassen 3 und 4 umgesetzt werden kann. Das Bildungsangebot wurde im Rahmen des von der Telekom-Stiftung geförderten Projekts „Die Zukunft des MINT-Lernens" entwickelt. Die Handreichung wurde im Rahmen des vom Bundesministerium für Bildung und Forschung geförderten MoSAiK-Projektes („Qualitätsoffensive Lehrerbildung") erstellt. Die Handreichung soll Grundschullehrkräfte dabei unterstützen, Schüler*innen Kompetenzen von Computational Thinking mit Hilfe des Mini-Computers von Calliope® und der Programmiersprache NEPO®, welche auf Open Roberta Lab[1] verfügbar ist, näherzubringen.

Computational Thinking ist ein Teil des MINT-Unterrichts (vgl. Shute et al., 2017) und wird deshalb in diesem Handbuch mit naturwissenschaftlich-technischen Denk-, Arbeits- und Handlungsweisen verknüpft. Denk-, Arbeits- und Handlungsweisen wie Vermuten, Ordnen, Beobachten und Bewerten sind nicht nur beim Experimentieren oder technischen Problemlösen elementare Kompetenzen, sondern können auch zur Unterstützung des Computational Thinking eingesetzt werden. So sollen Schüler*innen beispielsweise vermuten, welches geeignete Befehle zur Lösung eines Problems sein können; Befehle & Befehlsabfolgen ordnen; beobachten und beschreiben, wie sich verschiedene Programmcodes bei der Ausführung unterscheiden; die gefundenen Unterschiede anhand des Programmcodes erklären und bewerten.

In dieser Handreichung finden Sie zunächst einen Überblick zum Bildungsangebot und zu Calliope mini®, organisatorische Hinweise sowie Erläuterungen zu zusammenfassenden Karteikarten, die neben Aufgaben und Musterlösungen auch Impulse für die Unterstützung der Schüler*innen im Unterricht enthalten (Kapitel 2). Zudem erhalten Sie einen kurzen Einblick in den fachlichen Hintergrund zu sechs wichtigen Kompetenzen des Computational Thinking: „Abstrahieren", Generalisieren", „Zerlegen in Teilprobleme", „Sequenzieren von Algorithmen", „Kontrollfluss" und „Debugging" (Kapitel 3). Der Hauptteil der Handreichung enthält nach einer Einführung zur Kompetenzorientierung und dem didaktischen Ansatz des angeleiteten forschenden Lernens eine Übersicht zu sechs Sequenzen des Bildungsangebots mit ihren Inhalten und den angestrebten Kompetenzen bei Grundschulkindern sowie jeweils eine Kurzbeschreibung und konkrete Hinweise zur Durchführung (Kapitel 4). Schließlich werden in der Handreichung Anregungen für eine adaptive Förderung von Schüler*innen im Rahmen des Bildungsangebotes gegeben (Kapitel 5) und Planungshilfen hierfür bereitgestellt (Kapitel 6).

Diese Handreichung hat die Zielsetzung, die Bildungsarbeit im Bildungsbereich „Computational Thinking als 21st Century Skill" zu unterstützen. Sie zeichnet sich durch die

[1] Roberta, Open Roberta und NEPO sind eingetragene Warenzeichen der Fraunhofer-Gesellschaft e.V. Die Programmierumgebung ist erreichbar unter lab.open-roberta.org. Weitere Informationen zur Roberta-Initiative von Fraunhofer IAIS finden sich unter www.roberta-home.de.

Verwendung problemorientierter Aufgaben und den Fokus auf die Kompetenzen des Computational Thinking aus und enthält daher keine kleinschrittigen Lehrgänge oder „Programmierrezepte" zum Nachahmen.

Auch Lehrkräfte können sich über das hier beschriebene problemorientierte Bildungsangebot mit dem Calliope und seiner Programmierung vertraut machen. Es werden lediglich Grundkenntnisse in der Bedienung eines PCs oder Laptops benötigt. Wer zunächst die Programmiersprache NEPO® kennenlernen möchte, kann als ersten Schritt die Karteikarten anschauen und einige oder alle Beispielaufgaben in der Programmierumgebung auf Open Roberta Lab ausprobieren. Weitere Programmierideen finden sich im Internet (z.B. auf YouTube) und in zahlreichen Veröffentlichungen zum Programmieren mit dem Calliope mini®, (bspw. bei Abend et al., 2017; Bergner et al., 2017; Bergner & Leonhardt, 2018; Bockermann et al., 2018; Kiefer, 2018; Körner & Bettner, 2021)[2]. Für das Programmieren mit NEPO® auf Open Roberta Lab wird kein Calliope benötigt, da in der Programmierumgebung ein Calliope-Simulator enthalten ist.

Um einen problemorientierten und kompetenzorientierten Unterricht zum Programmieren zu ermöglichen, sollten nach einer ersten Erkundung der Programmierumgebung die Erläuterungen zum Bildungsangebot in diesem Handbuch studiert werden. Mit diesem Wissen können auch Aufgaben aus anderen Veröffentlichungen im Hinblick auf ihre Problemorientierung und Kompetenzorientierung beurteilt und ggf. angepasst werden[3].

[2] Einige der genannten Veröffentlichungen nutzen andere Programmierumgebungen („Editoren"), die aber wie NEPO® blockbasiert sind. I.d.R. enthalten auch die anderen Editoren einen Simulator und können zunächst ohne Calliope mini® ausprobiert werden.

[3] Wer nach dem Studieren des Handbuches dennoch eine explizite Einführung in die Bestandteile und Handhabung des Calliope mini® oder der Programmiersprache NEPO® wünscht, kann z.B. bei Abend et al. (2017) oder https://calliope.cc/los-geht-s/erste-schritte nachlesen. Bei Bockermann et al. (2018) finden Sie auch Anleitungen für Kinder, z.B. dazu wie ein Programm auf den Calliope mini® geladen werden kann.

2. Das Bildungsangebot: Überblick und Organisatorisches

Das in diesem Handbuch beschriebene Bildungsanbot eignet sich besonders für Kinder im Alter von acht bis zehn Jahren. Es enthält sechs problemorientierte Aufgaben zum Thema Computational Thinking. Grundschulkinder können alltagsnahe Erfahrungen mit Aspekten des Programmierens machen und ein erstes Verständnis dafür entwickeln, welche Bedeutung Programmieren für das Funktionieren eines computerbasierten Gegenstandes hat. Darüber hinaus lernen sie Computational Thinking – also computerbezogene Kompetenzen wie „Abstrahieren", „Generalisieren", „Zerlegen in Teilprobleme", „Sequenzieren von Algorithmen", „Kontrollfluss" und „Debugging".

Nach unserer Erfahrung bietet es sich an, das Bildungsangebot im Gruppenunterricht durchzuführen. Dies ist besonders gewinnbringend, da Grundschulkinder unterschiedliche Zugangsweisen zu einer Programmieraufgabe haben und unterschiedliche Lösungsstrategien entwickeln. So können die Schüler*innen mit- und voneinander lernen. Generell kann das Bildungsangebot aber auch mit weniger oder mehr Schüler*innen durchgeführt werden.

Das hier vorgestellte Bildungsangebot soll dabei unterstützen, Unterricht in der eigenen Klasse zu planen und durchzuführen. Im Optimalfall wird eine problemorientierte Aufgabe des Bildungsangebots während einer Doppelstunde eingesetzt. Die Schüler*innen sollen eigenständig oder kollaborativ Lösungen entwickeln und umsetzen, und sollen bei Bedarf dabei von der Lehrkraft unterstützt werden. Dabei ist zu beachten, dass das Programmieren immer ein Ausgangspunkt sein soll, andere Lösungen auszuprobieren, diese miteinander zu vergleichen und / oder weitere Lösungen zu generieren. Durch das Erarbeiten und Vergleichen verschiedener Lösungsansätze zur gleichen Aufgabe wird ein tieferes Verständnis des Programmierens begünstigt. Bei einem reinen Ausprobieren „bis es funktioniert" wird dieses tiefere Verständnis oft nur bei einzelnen, sehr interessierten Schüler*innen erreicht, während weniger interessierte oder leistungsschwächere Schüler*innen evtl. die Suche nach einer funktionierenden Lösung abbrechen und / oder einfach eine evtl. vorhandene Musterlösung übernehmen, ohne sie wirklich zu durchdenken. Und auch Schüler*innen, die sich nicht trauen auszuprobieren, können durch das Vergleichen von Lösungsansätzen unterstützt werden.

Das aus fachlicher Sicht wichtige Ziel der möglichst sparsamen Programmierung kann von Grundschulkindern noch nicht erreicht werden. Es kann in der Grundschule vorbereitet und in der weiterführenden Schule in den Blick genommen werden. Zwar könnten in der Grundschule besonders elegante Musterlösungen präsentiert werden, dies würde aber nicht zum tieferen Verständnis des Programmierens beitragen, welches für das spätere eigenständige Finden von eleganten Lösungen erforderlich ist. In der Grundschule ist daher das Durchdenken und Verstehen von Programmieraufgaben, Lösungsprozessen und möglichen Lösungen zentral. Das Vergleichen und bewerten von unterschiedlichen Lösungen der Schüler*innen kann diesen

Prozess weiter vertiefen – selbst wenn dabei nicht die eleganteste Lösung gefunden wird. Beim Vergleichen und Bewerten der unterschiedlichen von den Schüler*innen gefundenen Lösungsmöglichkeiten kann das Kriterium der möglichst sparsamen Programmierung eingeführt und einbezogen werden: Die Programmcodes werden dann nicht nur dahingehend verglichen, welche unterschiedlichen Lösungswege möglich sind, sondern z.B. auch hinsichtlich der benötigten Anzahl der Befehle. Auf diesem Weg kann das Verständnis angebahnt werden, dass das Finden sparsamer Programmierlösungen ein wichtiger Aspekt des CT ist. Für dieses Verständnis muss aber nicht die allersparsamste Lösung gefunden werden.

Die Lehrkraft unterstützt das Finden und Vergleichen von Lösungen mit gezielten Hilfestellungen (vgl. Kapitel 5), gibt aber keine vorab definierten Lösungsschritte vor, die von den Schüler*innen stur abzuarbeiten wären (vgl. Kapitel 4). Sie gibt den Schüler*innen Werkzeuge an die Hand, mit deren Hilfe sie sich selbst in das Problem hineindenken, eigene Lösungsideen planen und aufschreiben und sich danach gegenseitig austauschen, bevor sie einzelne Lösungen programmieren (vgl. Kapitel 4 und 5). Zum Schluss einer Problemlöse-aufgabe können dann die Schüler*innen wieder zusammenkommen, um die unterschiedlichen Zugangsweisen in ihren Gruppen der ganzen Klasse vorzustellen. Dieses didaktische Vorgehen entspricht dem didaktischen Ansatz des angeleiteten forschenden Lernens im naturwissenschaftlichen Unterricht (vgl. Kapitel 4).

Aufbau einer Doppelstunde

Eine Doppelstunde, in der eine Aufgabe bearbeitet wird, könnte so ablaufen:

Einstieg	Die Problemstellung wird der Klasse vorgestellt.
Einzelarbeit	Schüler*innen planen mit Papier und Bleistift die Schritte, die sie für die Lösung des Problems gehen müssen. Dafür haben Sie eine Übersicht über die einzelnen Programmierbausteine zur Verfügung.
Gruppenarbeit 3-5 Schüler*innen	Die Schüler*innen kommen in den Gruppen zusammen und tauschen sich über ihre Planungen aus: • Die unterschiedlichen Lösungsansätze werden ausprobiert bzw. programmiert. • Das Verhalten des Calliope wird festgehalten und überprüft, ob das Programmierziel erreicht wurde. • Verschiedene Lösungsansätze werden verglichen und für Hinweise auf Programmierfehler und alternative Lösungswege genutzt. • Die Lösungsansätze werden verbessert. Die einzelnen Schritte werden protokolliert und die Schüler*innen erstellen einen Gruppenbericht für den Austausch im Klassengespräch.
Reflexion	Die Schüler*innen kommen in der gesamten Klasse zusammen und tauschen ihre Erfahrungen aus.

Im Anschluss haben die Schüler*innen häufig eigene Ideen, was sie als nächstes Programmieren möchten. Insofern sind die hier vorgestellten problemorientierten Aufgaben als Ausgangspunkte zu verstehen, in deren Anschluss die angestrebten Kompetenzen mit weiteren, von den Schüler*innen erfundenen Aufgaben bearbeitet werden. Aus didaktischer Sicht ist es für das Lernen zentral, dass die Lehrkraft die in Kapitel 3 und 4 beschriebenen anzustrebenden Kompetenzen des Computational Thinking sowie die Lernwege der Schüler*innen im Blick behält. Dies kann gelingen, wenn sie die Lernstände der Schüler*innen kontinuierlich diagnostiziert, ihnen nur genau so viel Hilfe gibt, wie notwendig ist, und ausgehend vom Erreichten weitere Lernschritte plant (Criblez et al., 2009; Vygotsky, 1978). Prinzipiell können die gleichen Aufgaben sowohl zum Kompetenzaufbau als auch zur Kompetenzüberprüfung eingesetzt werden (vgl. Kapitel 5). Sowohl Aufbau als auch Überprüfung von Kompetenzen erfordern kompetenzorientierte und problemorientierte Aufgaben (Criblez et al., 2009; Reusser, 2005).
Neben Unterstützungsmaßnahmen in Bezug auf die Kompetenzen des Computational Thinking und die Denk-, Arbeits- und Handlungsweisen können auch sprachliche Unterstützungsmaßnahmen erforderlich sein, da die Schüler*innen zur Kommunikation über ihre Lösungsansätze bestimmte Begrifflichkeiten und Satzbausteine benötigen (Gibbons, 2006; vgl. Kapitel 5).

Aufbau der Karteikarten

Die Karteikarten bestehen jeweils aus einer Vorder- und Rückseite und enthalten u.a. Informationen zu den in der jeweiligen Sequenz angestrebten Teilkompetenzen des Computational Thinking und Impulse, wie diese problemorientiert im Unterricht entwickelt und gefördert werden können:

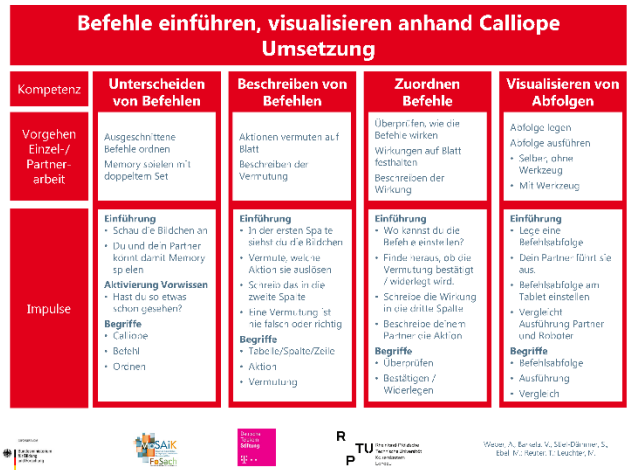
<table>
<tr><td colspan="2">Vorderseite</td></tr>
<tr><td>Titel: Aktionen, die Lehrkräfte initiieren, um, Kompetenzen des Computational Thinking bei Schüler*innen zu fördern.

Erste Spalte: In der problemorientierten Aufgabe angestrebte Teilkompetenzen (hier 4 Stück).

Zweite Spalte: Das Wissen, das Lehrkräfte dazu benötigen

Dritte Spalte: Die Materialien, die zur Bearbeitung der Aufgabe eingesetzt werden.</td><td> 
</td></tr>
<tr><td colspan="2">Rückseite</td></tr>
<tr><td>Titel: Angestrebte Kompetenz des Computational Thinking

Spalten mit den Teilkompetenzen (entsprechend der ersten Spalte der Vorderseite, hier also vier)

Impulse: Zu jeder Teilkompetenz sind Ideen zur Vorgehensweise skizziert und darunter mögliche Anregungen und Unterstützungsmaßnahmen, die von den Lehrkräften gegeben werden können sowie Begriffe, die angewendet werden können. Diese sind nicht abschließend dargestellt.</td><td> 
</td></tr>
</table>

3. Fachlicher Hintergrund

In einer zunehmend digitalisierten Welt kommen Kinder bereits früh mit Medien in Kontakt und nutzen programmierte Geräte wie Smartphones und Computer, verstehen aber nicht wie sie funktionieren (Feierabend, S., Rathgeb, T., and Reutter, T., 2019). In verschiedenen internationalen Studien konnte gezeigt werden, dass Kindern das Programmieren in Grundzügen schon im Kindergarten- und Grundschulalter nähergebracht werden kann (Di Lieto et al., 2017; Dickes et al., 2016). Die Kompetenzen, die für das Programmieren benötigt werden, werden im fachdidaktischen Kontext des digitalen Lernens als Computational Thinking bezeichnet und sind ein wichtiger Baustein des MINT-Lernens (vgl. Shute et al., 2017; A. M. Weber et al., 2022). Für einen solchen Unterricht benötigen Grundschullehrkräfte u.a. Fachwissen über das Programmieren und spezifisches fachdidaktisches Wissen zu Computational Thinking (Angeli et al., 2016; Paniagua & Istance, 2018; A. M. Weber et al., 2022). Nach Wing (2006) bezieht sich der Begriff Computational Thinking (CT) auf einen Denkprozess, mit dem Lösungen für Probleme so dargestellt werden, dass sie von einem Informationsverarbeiter verstanden werden können. Der Informationsverarbeiter kann, muss jedoch kein Computer sein. CT entspricht in weiten Teilen dem informatischen problemlösenden Denken, das z. B. im Schreiben von Programmen genutzt werden kann (A. Weber et al., 2021).

In der Psychologie spricht man von einem Problem, wenn ein unerwünschter Ausgangszustand in einen gewünschten Zielzustand überführt werden soll, dieser Zielzustand jedoch nicht unmittelbar zu erreichen ist. Zwischen dem Erreichen des Ziels und dem Ausgangszustand besteht ein Hindernis – zum Beispiel fehlen dem Problemlösenden routinierte Prozeduren, um vom Ausgangs- zum Zielzustand zu gelangen (Funke, 2003). Probleme können mehr oder weniger komplex sein, je nach Anzahl der zu beachtenden Variablen und deren Vernetztheit sowie hinsichtlich der Problemtransparenz und -klarheit. Bei einem wohldefinierten Problem (hohe Transparenz und Klarheit) sind Ausgangslage, Zielsetzung und Barrieren, die der Zielerreichung entgegenstehen, eindeutig definiert (Funke, 2003). Bei schlecht definierten Problemen (geringe oder keine Transparenz und Klarheit) ist dies nicht der Fall. Aus didaktischer Sicht gilt es darauf zu achten, dass die im Lernangebot behandelten Probleme für Grundschulkinder zwar heraus-, aber nicht überfordernd sind. Vor diesem Hintergrund stehen wohldefinierte Probleme im Mittelpunkt dieses Bildungsangebots (vgl. Kapitel 4).

Programmieren als Problemlöseprozess

Computational Thinking beim Programmieren dient dem Lösen informatischer Probleme und weist einige Parallelen mit dem technischen Problemlösen (vgl. z.B. Lucas et al., 2014) auf. Bei beiden Problemlöseprozessen muss zunächst das Problem erkannt und verstanden werden, um anschließend Lösungsideen zu generieren, einen Lösungsprozess zu planen, eine mögliche Lösung zu erarbeiten und zu testen. Das Testergebnis wird bewertet und es werden ggf. Optimierungen am Modell bzw. Programm vorgenommen. Dieser Prozess ist **systematisch**, da er einer charakteristischen Sequenz aus (1) Probleme identifizieren, (2) Lösungen planen, (3) Modelle / Prototypen / Programme erstellen und (4) Lösungen testen, bewerten und optimieren folgt (vgl. Lucas et al., 2014).

Der Engineering-Design-Prozess ist ebenso wie das informatische Problemlösen **iterativ**, da Ingenieur:innen ebenso wie Programmierer:innen verschiedene Lösungsentwürfe planen und in Prototypen bzw. Programme umsetzen, ihre Lösungen kontinuierlich testen und optimieren, Daten analysieren und interpretieren, aus Fehlern lernen und Entscheidungen auf der Grundlage von Testergebnissen treffen, mit dem Ziel, die Lösungsqualität stetig zu verbessern (Lucas et al., 2014).

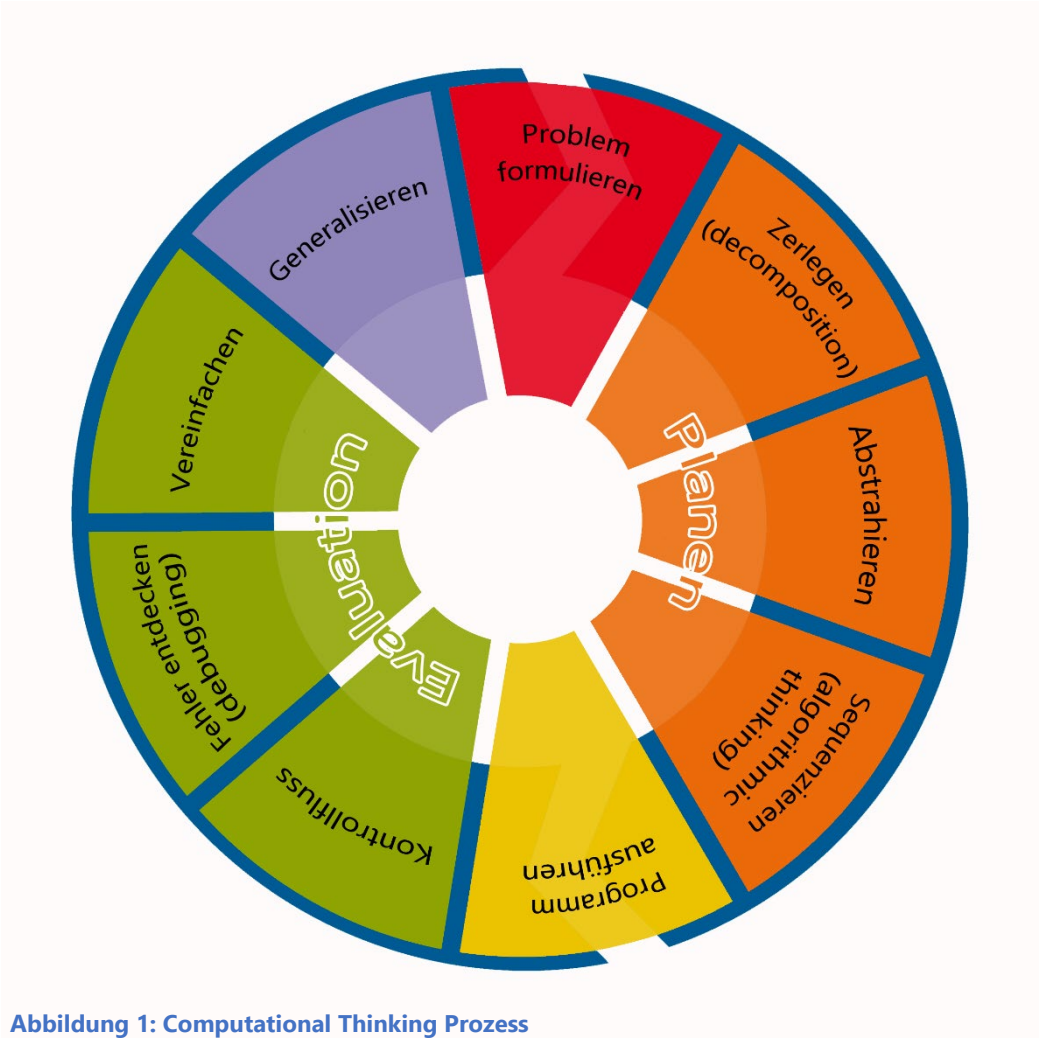


Abbildung 1: Computational Thinking Prozess

Beim Computational Thinking nach Angeli et al. (2016) umfasst das Planen u.a. das Zerlegen, Abstrahieren und Sequenzieren der Problemstellung (vgl. Abb. 1), welches auch das Erstellen des Programmcodes beinhaltet. Das Ausführen des Programmcodes ist die Voraussetzung für die anschließende Evaluation. Das Testen und Bewerten besteht beim Programmieren aus dem Ausführen des Programmes und dem Kontrollfluss. Das Optimieren umfasst beim Programmieren Debugging und Vereinfachung. Generalisieren ermöglicht die Übertragung der gefundenen Lösung auf weitere Programmierprobleme (vgl. auch A. M. Weber et al., 2022). Nachfolgend werden die einzelnen Schritte des in Abb. 1 dargestellten Computational Thinking Prozesses näher beschrieben. Zum besseren Verständnis werden sie an der Aufgabe „Der Calliope mini® als automatisches Fahrradrücklicht“ aus dem Lehrermaterial „Coden mit dem Calliope mini®“ (Abend et al., 2017) und daran angelehnten selbstentwickelten Aufgaben illustriert.

Planen: Abstrahieren, Zerlegen, Sequenzieren

Abstrahieren

Abstrakte Probleme werden durch konkrete Aufgaben veranschaulicht. Die Aufgabe in Sequenz 4 „das Fahrradlicht soll leuchten, wenn es dunkel wird“ (vgl. Abend et al., 2017) ist ein Beispiel für das abstrakte Problem „wenn Lichtsensor unter einer bestimmten Zahl, dann Aktion“. Um diese Aufgabe zu lösen, ist es notwendig herauszufinden, wie eine logische Abfolge von Befehl und Aktion mit Hilfe eines Informationsverarbeiters umzusetzen ist (vgl. S. 18, 23 und 73 – 75 dieser Handreichung).

Aufgaben werden abstrahiert, indem relevante Eigenschaften verallgemeinert formuliert werden und irrelevante Eigenschaften ausgeschlossen werden. Zur Programmierung des Fahrradlichts muss bspw. als relevant erkannt werden, dass die Helligkeit gemessen werden muss und dass ein Licht leuchten soll, solange ein gewisser Helligkeitswert unterschritten ist. Irrelevant (und in der Aufgabenstellung auch nicht genau definiert) ist bspw. ob die RGB-LED oder das 5x5-LED-Display des Calliope mini® als Licht genutzt werden.

Durch Abstraktion kann eine passende Repräsentation des Problems erarbeitet (Wing, 2006), Komplexität reduziert (Angeli et al., 2016), und das eigentliche Problem fokussiert werden (Curzon et al., 2014). Abstrahieren ist (zusammen mit Zerlegen und Sequenzieren) notwendig, um die hinter einer Aufgabe liegenden Algorithmen aufzudecken und zu programmieren.

Zerlegen (Decomposition)

Komplexe Aufgaben werden in Teilprobleme zerlegt. Teilprobleme werden dabei so aufgegliedert, dass diese entweder leichter zu verstehen und lösen sind oder dass Lösungsstrategien bereits bekannt sind (Faber et al., 2017). Anschließend können, wie bei einem Puzzle, mehrere Teilprobleme zusammengesetzt werden (vgl. Sequenzieren). Um bspw. die Fahrradlicht-Aufgabe aus Sequenz 4 leichter verstehen und lösen zu können, kann die

Aufgabe in die Teilprobleme „erkennen, wenn es dunkel ist", „Licht anmachen", „Wenn-Dann-Bedingung" und „unendlich oft machen" zerlegt werden. Für jedes Teilproblem kann dann überlegt werden, wie der passende Programmcode aussehen könnte:

- Calliope mini® kann mit einem der Aktionsbausteine „Schalte LED an Farbe + Farbbaustein" oder „Zeige Bild + Bild-Baustein" Licht einschalten. Man kann also schon einmal den benötigten Baustein bereitlegen.
- Genauso kann man an das Teilproblem „erkennen, wenn es dunkel ist" herangehen: Calliope mini® kann mit dem Lichtsensor erkennen, wie hell es ist. Man benötigt also den Baustein „Gib Wert % Lichtsensor" aus dem Sensor-Bereich.
  Damit Calliope mini® überprüfen kann, ob es dunkel ist, muss der aktuelle Sensorwert mit einem Soll-Wert abgeglichen werden. Der Soll-Wert wird mit dem Zahlen-Baustein aus dem Mathematik-Bereich festgelegt, der Abgleich erfordert den Logik-Baustein, der zwei Werte miteinander vergleicht. Das Teilproblem „Erkennen, wenn es dunkel ist" erfordert also eine Kombination aus drei Bausteinen.
- Die Wenn-Dann-Bedingung erfordert den Kontroll-Baustein „wenn mache". Es ist aber auch möglich, eine funktionierende Lösung mit „wenn mache sonst" zu programmieren und für die sonst-Bedingung z.B. den Aktionsbaustein „Schalte LED aus" einzufügen.
- Der Kontrollbaustein „Wiederhole unendlich oft" ist notwendig, damit Calliope mini® beständig Ist- und Sollwert vergleicht und das Licht entsprechend automatisch an- und ausschaltet.

Zerlegen kann aber nicht nur bei der Programmierung eines „einfachen" automatischen Fahrradlichtes helfen, sondern auch bei der Lösung der komplexen Fahrradlicht-Aufgabe in Sequenz 6 „Erstelle ein automatisches Fahrradlicht, bei dem ausgewählt werden kann, ob die Lampen durchgehend leuchten oder blinken". Diese kann in drei Teilprobleme zerlegt werden, sodass die entstehenden Teilprobleme mit Hilfe der in den vorherigen Sequenzen erworbenen Kompetenzen gelöst werden können (vgl. S. 19, 25 und 80 – 83):

1) Eine Befehlsabfolge für „die Lampe leuchtet durchgehend".
2) Eine Befehlsabfolge für „die Lampe blinkt".
3) Erstellen des Menüs, in dem „durchgehend leuchten" oder „blinken" eingestellt werden kann.

Das Zerlegen in Teilprobleme ist relevant, um die Komplexität des Problems zu reduzieren und die Lösung schrittweise erarbeiten zu können (vgl. Angeli et al., 2016). Wie Schüler*innen dabei unterstützt werden können, ohne die oben beschriebenen (Teil-)Lösungen vorzugeben, ist in Kapitel 5 und 6 erläutert.

Sequenzieren (Algorithmic Thinking)

Durch das Sequenzieren von Algorithmen (vgl. Sequenz 2 auf S. 17, 21 und 64 – 67) kann die zentrale Erkenntnis angebahnt werden, dass ein Computer nicht fähig ist, selbständig zu denken. Jeder Schritt einer gewünschten Aktion muss detailliert im Computerprogramm definiert werden. Eine Schwierigkeit dabei ist, zu erkennen, aus welchen Schritten ein Algorithmus besteht (vgl. Zerlegen) und in welcher Reihenfolge diese Schritte angeordnet werden müssen (Angeli et al., 2016; Curzon et al., 2014). Jedoch weisen Zhang und Nouri (2019) in ihrem Review darauf hin, dass mit Hilfe von Analogien aus dem täglichen Leben, wie z. B. To-Do-Listen oder Bedienungsanleitungen das Sequenzieren für Schüler*innen und Lehrkräfte einfach zu lernen sei (vgl. auch Relkin & Strawhacker, 2021). Die einzelnen Teilprobleme und Programmierbausteine der Aufgabe „das Fahrradlicht soll leuchten, wenn es dunkel wird" wurden bereits unter „Zerlegen" erläutert. Beim Sequenzieren werden diese Teilschritte nun in die richtige Reihenfolge gebracht:

1) Überprüfe, ob der Lichtsensor unter Wert x (z.B. 30%) liegt).
2) Wenn Wert des Lichtsensors unterhalb 30% Lichtstärke ist, lasse Lampe leuchten
4) trifft 3) nicht zu, leuchtet Lampe nicht,
4) wiederhole die Schritte 1) – 3) unendlich oft.

Beim Programmieren der Teilschritte in der o.g. Reihenfolge wird zudem in der blockbasierten Programmiersprache NEPO® besonders deutlich sichtbar, dass die Schleife („wiederhole die Schritte 1) – 3) unendlich oft") eigentlich zuerst geschrieben werden muss, da sie alle vorherigen Schritte umschließt. Man muss alle für Schritte 1) – 3) angeordneten Blöcke im letzten Schritt in den „wiederhole unendlich oft"-Baustein hineinziehen.
Das Sequenzieren eines Algorithmus wird also benötigt, um das Programm fehlerfrei verfassen zu können (Angeli et al., 2016).

Evaluation: Kontrollfluss, Debugging und Vereinfachung

Die Evaluation eines Algorithmus kann als dreiteilig verstanden werden: Kontrollfluss, Debugging und Vereinfachung.

Kontrollfluss

Damit ein Algorithmus die Aufgabe wie gewünscht ausführen kann, ist es nicht nur notwendig, jeden Schritt zu identifizieren und zu definieren, sondern diese auch in der richtigen Reihenfolge anzuordnen. Wurde zu Beginn des Problemlösungsprozesses die mögliche Lösung der Aufgabe geplant und aufgezeichnet, so kann nun das entstandene Programm mit dem Plan verglichen werden. Durch die Evaluation können Konzeptionsfehler sichtbar gemacht und alternative Lösungen verglichen werden (Faber et al., 2017). Bei der oben genannten Aufgabe wird also überprüft, ob die geplanten Schritte in dieser Reihenfolge ablaufen können. Bei der Planung des Problems wäre bspw. möglich, dass zunächst mit dem offensichtlichen Befehl

„Schalte die Lampe an" begonnen wird, dann die Bedingung und deren Überprüfung formuliert werden und dann „wiederhole unendlich oft" hinzugefügt wird. In dieser Reihenfolge würde das Programm allerdings nicht korrekt laufen.

Debugging:

Anschließend wird der Algorithmus ausgeführt und überprüft, ob die gewünschten Aktionen ausgeführt wurden. Debugging bedeutet, dass bei einem fehlerhaften Algorithmus Fehler identifiziert und beseitigt werden. Ein sicherer Umgang mit den einzelnen Befehlsbausteinen ist dabei ausschlaggebend. Oftmals sind kleine Abweichungen oder Fehlkonzeptionen verantwortlich für einen fehlerhaften Algorithmus. Ein gängiger Fehler bei der Aufgabe „das Fahrradlicht soll leuchten, wenn es dunkel wird" ist, dass vergessen wurde, den Befehl „wiederhole unendlich oft" zu programmieren. So würde der Informationsverarbeiter den Befehl nur einmal verarbeiten und die Lampe würde (je nach Umgebungshelligkeit und im Programm definiertem Grenzwert) gar nicht leuchten oder nur einmal für einen extrem kurzen Moment. In Sequenz 3 wird das Debugging anhand von fehlerhaften Programmcodes einer Hupe erarbeitet, bei denen u.a. dieser häufig von Schüler*innen gemachte Fehler gefunden werden muss (vgl. S. 17, 22 und 68 – 72).

Vereinfachung

Während der Evaluation liegt das Augenmerk nicht nur auf einem korrekt geschriebenen Algorithmus, sondern auch auf der Effizienz und somit Vereinfachung (Curzon et al., 2014), d.h. ein Programm sollte so wenig Schritte wie möglich benötigen, um die Aktionen auszuführen (Faber et al., 2017). Zum Beispiel muss bei der Fahrradlicht-Aufgabe geklärt werden, ob der Informationsverarbeiter den Befehl „schalte Licht aus" benötigt oder ob das Licht automatisch ausgeht, wenn die Bedingung nicht mehr erfüllt ist. Die drei Schritte Kontrollfluss, Debugging und Vereinfachung sind also für die Evaluation von eleganten Lösungen informatischer Probleme unabdingbar (Angeli et al., 2016). Da das eigenständige Finden besonders effizienter Lösungen in der Grundschule unrealistisch ist, werden Vereinfachungsmöglichkeiten auf Basis des Vergleiches der verschiedenen Lösungen der Schüler*innen erarbeitet. Solche Vergleiche sind prinzipiell in allen Sequenzen möglich.

Generalisieren

Am Ende des informatischen Problemlösezyklus steht das Generalisieren. Gefundene Lösungen werden allgemein formuliert und auf weitere Aufgabenstellungen mit gleichen Eigenschaften übertragen (Selby, 2014). Wenn Aufgabenstellungen auf das zugrundeliegende Problem abstrahiert werden können, gelingt es leichter, auch in anderen Aufgabenstellungen dieses abstrakte Problem zu entdecken. So können Probleme leichter präzisiert sowie Lösungsstrategien synergetisch genutzt werden (Curzon et al., 2014). In unserem Beispiel kann der Algorithmus nicht nur für das Programmieren eines Fahrradlichts verwendet, sondern auch auf

einen Bewegungsmelder oder eine Alarmanlage für ein Kinderzimmer übertragen werden. Generalisieren ist demnach für ein flexibles Lösen von verschiedenen informatischen Problemen relevant (Angeli et al., 2016). In Sequenz 5 wird das Generalisieren an einem sehr einfachen Beispiel eingeführt: Es werden verschiedene Programmcodes gezeigt und analysiert, die ein „H" auf dem Bildschirm des Calliope mini® erscheinen lassen (vgl. S. 18, 24 und 76 – 79).

4. Die Unterrichtssequenzen

Angestrebte Kompetenzen

Der Aufbau der übergeordneten Kompetenz „Computational Thinking als 21st Century Skill" ist als längerfristiger kumulativer Prozess zu sehen (Criblez et al., 2009). Das in diesem Handbuch beschriebene Lernangebot zum Calliope mini® hat das Ziel, die Grundprinzipien des Computational Thinking an einem überschaubaren Beispiel zu erlernen. Auf diesen Grundlagen kann in weiteren Lehr-Lern-Prozessen aufgebaut werden, indem Computational Thinking auf weitere Anwendungsmöglichkeiten (z.B. Lego WeDo oder BlueBot oder auch auf das Programmieren mit textuellen Programmiersprachen) übertragen wird. Hierfür müssen zwar spezifisches Wissen und Fertigkeiten bzgl. der neuen Anwendung erlernt werden (z.B. veränderte Funktionen oder eine andere Programmiersprache), es können aber die bereits erworbenen Kompetenzen des Computational Thinking – wie „Abstrahieren", „Generalisieren", „Zerlegen in Teilprobleme", „Sequenzieren von Algorithmen", „Kontrollfluss" und „Debugging" – eingesetzt und weiter ausgebaut werden. Aus diesem Grund steht die Förderung dieser Kompetenzen im Fokus dieses Handbuchs.

Didaktischer Ansatz

Die didaktische Umsetzung des vorliegenden Bildungsangebots folgt dem Ansatz des angeleiteten forschenden Lernens im naturwissenschaftlichen Unterricht, dessen Effektivität durch zahlreiche Studien wissenschaftlich sehr gut abgesichert ist (Furtak et al., 2012; Lazonder & Harmsen, 2016; Reuter & Leuchter, 2019). Die dabei zum Einsatz kommenden problemorientierten Aufgaben fordern physische und mentale Aktivität („Hands-on – Minds-on") der Schüler*innen. Beim angeleiteten forschenden Lernen arbeiten die Schüler*innen eigenständig – jedoch mit Unterstützung der Lehrkraft (vgl. Kapitel 5). Der Einsatz kognitiv aktivierender verbaler Unterstützungsmaßnahmen hängt mit einem stärkeren Lernzuwachs beim naturwissenschaftlich-technischen Lernen zusammen (Ebel, 2020; Hofer et al., 2018; Schmitt et al., 2023).

Aus didaktischer Sicht sind für Grundschulkinder wohldefinierte problemorientierte Aufgaben besser geeignet als offen formulierte Probleme (vgl. z.B. Crismond, 2001). Das Bildungsangebot setzt daher auf Probleme, die ein klares Ziel (z.B. „bringe ein Fahrradlicht zum Leuchten") und einen klaren Ausgangszustand (vorgegebenes Material, bekannte Programmierumgebung, …) haben. Zudem liegt die Barriere zwischen Ausgangs- und Zielzustand darin, dass prinzipiell bekannte Befehle in eine bestimmte (unbekannte) Anordnung gebracht werden müssen, nicht aber die Befehle – und damit die Mittel zur Problemlösung – völlig unbekannt sind. Wichtig ist jedoch, dass die Problemstellungen unterschiedliche Lösungen und Lösungswege zulassen sowie ein iteratives Vorgehen mittels Testen und Optimieren herausfordern (Crismond, 2001).

Durch die Vorgabe einer transparenten und klaren Zielstellung wird ein bloßes Explorieren verhindert. Um problemlösendes Denken anzuregen und zu fördern, ist es jedoch von zentraler Bedeutung, dass keine vorab festgelegten Lösungsschritte abgearbeitet werden (vgl. Grygier & Hartinger, 2012). Die Schüler*innen entwickeln eigenständig oder kollaborativ Lösungen und setzen diese entsprechend um.

Übersicht über Ziele und Inhalte der sechs Sequenzen

Sequenz 1 hat den Aufbau von Wissen zu Eigenschaften und Funktionsweise des Calliope mini® sowie das Kennenlernen der Programmiersprache als zentrale Fertigkeit für die Lösung von Programmierproblemen mit dem Calliope mini® zum Ziel. In den Sequenzen 2 bis 6 werden nacheinander die computerbezogenen Kompetenzen „Abstrahieren", „Generalisieren", „Zerlegen in Teilprobleme", „Sequenzieren von Algorithmen", „Kontrollfluss" und „Debugging" eingeführt. Wie bereits in Kapitel 3 erfolgt die Illustration des Kompetenzaufbaus in Sequenz 3, 4 und 6 am Beispiel der Aufgabe „Der Calliope mini® als automatisches Fahrradrücklicht" aus dem Lehrermaterial „Coden mit dem Calliope mini®" (Abend et al., 2017), wobei die Aufgabe je nach angestrebter Kompetenz in veränderter Form eingesetzt wird.

Für jede Sequenz werden Hinweise bzgl. der Unterstützung des Kompetenzerwerbs durch die Lehrkraft gegeben. In Kapitel 5 werden diese Unterstützungsmöglichkeiten sequenzübergreifend weiter ausgeführt.

SEQUENZ 1
Befehle einführen und visualisieren anhand Calliope mini®

ZIELE

Die Schüler*innen ...

- unterscheiden Befehle
- beschreiben Befehle als Aktionen
- ordnen Befehle zu Wirkungen zu
- visualisieren erste Abfolgen von Befehlen

INHALT

Kennenlernen von Befehlen und einfachen Befehlsabfolgen und ihren Wirkungen.

SEQUENZ 2
Algorithmen entwickeln anhand Calliope mini®

ZIELE

Die Schüler*innen ...

- erstellen und programmieren Befehlsabfolgen
- gruppieren Befehlsabfolgen und verstehen sie als Bausteine
- bringen Bausteine in Reihenfolgen (Sequenzieren)
- evaluieren die Reihenfolge der Bausteine (Kontrollfluss)

INHALT

Einfache Programmierziele festlegen, den Lösungsweg beschreiben und dazu passende Befehle suchen. Erkennen, dass bestimmte Gruppen von Befehlen immer wieder kehren und diese als Bausteine für komplexere Programmieraufgaben nutzen.

Abfolgen von Bausteinen programmieren, überprüfen, ob das Programmierziel erreicht wurde und die Abfolge ggf. korrigieren.

SEQUENZ 3
Fehler finden und beheben anhand Calliope mini®

ZIELE

Die Schüler*innen ...

- erkennen Fehler
- identifizieren den Ort des Fehlers
- identifizieren die Art des Fehlers
- beheben den Fehler

INHALT

In vorgegebenen fehlerhaften Programmen die Fehler identifizieren, analysieren und beheben (Debugging).

SEQUENZ 4
Abstrahieren anhand Calliope mini®

ZIELE

Die Schüler*innen ...

- verstehen die Problemstellung
- identifizieren relevante Bausteine
- halten die grundlegende Struktur eines Problems fest
- minimieren die Komplexität

INHALT

Ein „automatisches Fahrradlicht" wird gezeigt und soll programmiert werden. Dafür passende Bausteine werden ausgewählt, ihre Anordnung geplant und die Funktionsweise anschließend überprüft und ggf. korrigiert. Es wird nach weiteren Lösungsmöglichkeiten gesucht und die „sparsamste" ausgewählt.

SEQUENZ 5
Muster generalisieren anhand Calliope mini®

ZIELE

Die Schüler*innen ...

- erkennen Muster und Regelmäßigkeiten
- entwickeln Regeln und Modelle
- überprüfen Regeln und Modelle
- führen verschiedene Probleme auf ein gemeinsames zugrundeliegendes Muster zurück

INHALT

Verschiedene Bausteine in Ihrer Funktionsweise vergleichen, Gemeinsamkeiten entdecken und daraus Regeln formulieren. Die formulierten Regeln überprüfen und ggf. korrigieren.

Bausteine zu vorgegebenen Regeln finden bzw. kombinieren, überprüfen und ggf. korrigieren.

SEQUENZ 6
Probleme und Prozesse zerlegen anhand Calliope mini®

ZIELE

Die Schüler*innen ...

- erkennen Probleme
- formulieren Prozesse
- erkennen Teilziele
- programmieren Teilprozesse

INHALT

Eine komplexe Problemstellung verstehen, in Teilziele zerlegen und Lösungen für die Teilziele programmieren, überprüfen und ggf. korrigieren.

Die Lösungsteile zu einer Gesamtlösung zusammenfügen, die Funktionsweise überprüfen und die Abfolge der Bausteine ggf. korrigieren.

Kurzbeschreibungen und Hinweise für die Sequenzen

Nachfolgend werden die angestrebten Kompetenzen und Impulse jeder Sequenz näher beschrieben. *Begriffe*, mit denen Grundschulkinder Schwierigkeiten haben können, sind in der Beschreibung *kursiv* gedruckt. Grundschulkinder können aber auch mit anderen als den hier hervorgehobenen Begriffen Schwierigkeiten haben. Zudem können auch z.B. beim Formulieren von Vermutungen oder Begründungen Schwierigkeiten auftreten. Es ist wichtig, solche sprachlichen Schwierigkeiten zu beheben, da Denk-, Arbeits- und Handlungsweisen eng mit Kommunikation verbunden sind („Sprachhandlungen"; Vollmer & Thürmann, 2013) und Sprache daher essenziell für das Lernen von und mit Denk-, Arbeits- und Handlungsweisen ist. Hinweise zur Sprachförderung im Rahmen des Lernangebots zum Calliope mini® werden in Kapitel 5 gegeben.

SEQUENZ 1
Befehle einführen und visualisieren anhand Calliope mini®

Kurzbeschreibung - Zentraler Inhalt:

- Anhand von Bildkärtchen (vgl. Karteikarten zu Sequenz 1) lernen die Schüler*innen die *Befehle* kennen, mit denen sich der *Calliope mini®* steuern lässt. Sie schauen sich die Kärtchen in Partnerarbeit an und versuchen sie zu *ordnen*. Das genaue Anschauen kann durch Memory-Spielen mit einem doppelten Kärtchen-Satz unterhaltsamer gestaltet werden.
- Zunächst vermuten die Schüler*innen, welche Aktionen des Calliope mini® die jeweiligen Befehle bewirken und halten ihre *Vermutungen* in eigenen Worten in einer *Tabelle* fest (vgl. Karteikarten zu Sequenz 1).
- Anschließend programmieren sie ihre Befehle und beobachten am Calliope mini®, welche Wirkung die Befehle tatsächlich haben. Sie *überprüfen* so ihre Vermutungen. Die Beobachtungen können die Vermutungen *bestätigen* oder *widerlegen*.
- Nach dem Kennenlernen der Einzelbefehle und ihrer Wirkungen legen die Schüler*innen in Partnerarbeit erste *Befehlsabfolgen* und probieren diese aus. Sie *vergleichen*, welche Schritte ein Mensch mit diesen Befehlen *ausführen* würde und welche Aktionen Calliope mini® mit diesen Befehlen ausführt.

In dieser Sequenz geht es noch nicht um Computational Thinking im engeren Sinne, sondern um das Kennenlernen des Calliope mini® und seiner Steuerung durch Befehle. Mit Hilfe der Denk-, Arbeits- und Handlungsweisen „Vermuten", „Beobachten" und „Vergleichen" werden Vermutungen formuliert und überprüft. Dieses Kennenlernen ist notwendig, um in den darauffolgenden Sequenzen Programmierprobleme bearbeiten und dabei Computational Thinking erlernen zu können.

Hintergrundinformationen und Hinweise

Grundschulkinder neigen dazu, sich ihre Vermutungen nicht bewusst zu machen und direkt auszuprobieren. Daher ist es wichtig, die Schüler*innen dazu anzuhalten, ihre Vermutungen aufzuschreiben. Weiterhin ist es wichtig, dass Beobachtungen und Vermutungen separat aufgeschrieben werden, da Grundschulkinder häufig annehmen, dass nur das Ergebnis der Beobachtung bedeutsam ist und / oder Vermutungen, die sich nicht bestätigen, als Fehler erachten. Deshalb sollte mit den Schüler*innen thematisiert werden, dass es keine „falschen Vermutungen" gibt und dass es nicht schlimm ist, wenn sich die Vermutung nicht bestätigt. Sowohl die Vermutung als auch die Beobachtung sollen aufgeschrieben werden und erhalten bleiben, um den Entwicklungsprozess der Lösung nachvollziehen zu können.

Für die Durchführung der Sequenz 1
verwenden Sie bitte die Karteikarten „Sequenz 1".

SEQUENZ 2
Algorithmen entwickeln anhand Calliope mini®

Kurzbeschreibung - Zentraler Inhalt:

- Die Schüler*innen erarbeiten *Befehlsabfolgen* (Algorithmus erstellen durch Abstrahieren und Sequenzieren), die als Bausteine für komplexere Programmier-aufgaben genutzt werden können.
- Mehrere erarbeitete Befehlsbausteine werden von den Schüler*innen verglichen, damit sie erkennen können, dass bestimmte Befehlsabfolgen immer wieder für unterschiedliche Programmieraufgaben benötigt werden (Generalisieren). Beim Vergleichen von Befehlsabfolgen kann auch erkannt werden, welche *unterschiedlichen* Befehlsabfolgen bei der *Ausführung* das *gleiche* Resultat beim Calliope mini® bewirken.
- Beim Zusammenfügen der Bausteine zu komplexeren Programmen müssen die *Bausteine* so zusammengefügt werden, dass das Programm fehlerfrei läuft. Die Schüler*innen *planen* eine *Bausteinabfolge*, *beobachten*, ob sie zum gewünschten Ergebnis führt oder wo ggf. Abweichungen auftreten (Kontrollfluss). Sie versuchen, diese Abweichungen zu *korrigieren*, indem sie die Anordnung / Reihenfolge der entsprechenden Bausteine verändern (Debugging).

Hintergrundinformationen und Hinweise

Beim Sequenzieren ist ein häufiges Problem, dass nicht alle Schritte einzeln aufgeführt werden oder Schritte ganz vergessen werden.
Abstrahieren ist eine komplexe kognitive Leistung, zu der Grundschulkinder in der Lage sind, wenn man sie gezielt zum Denken anregt und zum Abstrahieren herausfordert. Es ist notwendig, um Lösungsalternativen und schließlich schlankere Lösungen finden zu können. Auch in dieser Sequenz ist es daher wichtig, die Schüler*innen zum Planen und Beobachten anzuregen.

Für die Durchführung der Sequenz 2
verwenden Sie bitte die Karteikarten „Sequenz 2“.

SEQUENZ 3
Fehler finden und beheben anhand Calliope mini®

Kurzbeschreibung - Zentraler Inhalt:

Den Schüler*innen wird erklärt, dass Calliope mini® bei Drücken der Taste A wie eine Hupe funktionieren soll und es wird Ihnen ein (fehlerhafter) Programmcode zum Ausprobieren gegeben. Nachdem von den Schüler*innen erkannt wurde, dass das *Programm* nicht zum o.g. *Ziel* führt, sollen Sie den *Fehler* suchen.

- Ein erster Schritt ist das Zerlegen des Programmes in *Bausteine* und die *Überprüfung*, was die einzelnen Bausteine bewirken. Dabei sollen die Schüler*innen zunächst auf dem Papier (ggf. mit Hilfe der Befehls-Kärtchen) überlegen, welche Bausteine welche Wirkung haben. Entsteht der Fehler dadurch, dass ein fehlerhafter oder unpassender Baustein eingefügt wurde? Dann könnte es helfen, diesen Baustein *auszutauschen*.
- Neben der *Wirkung* der einzelnen Bausteine muss auch die Reihenfolge der Bausteine analysiert werden. Entsteht der Fehler dadurch, dass die Bausteine nicht in der richtigen Reihenfolge aneinandergefügt wurden? Dann sollte der Calliope mini® mit veränderter Reihenfolge der Bausteine *programmiert* werden.
- Schließlich könnte der Fehler auch dadurch verursacht werden, dass ein Baustein fehlt. Dann muss der fehlende Baustein eingefügt werden, um den Fehler zu beheben.
- Mit verschiedenen fehlerhaften Programmcodes können verschiedene Szenarien für die Fehlersuche hergestellt werden. Auch eine Kombination mehrerer Fehlertypen ist prinzipiell möglich und kann als besondere Herausforderung zur Differenzierung eingesetzt werden.

Hintergrundinformationen und Hinweise

Es wichtig, dass die Schüler*innen dazu angehalten werden, Ihre Vermutungen über die Ursachen des Fehlers und die Lösungsmöglichkeiten vor dem Ausprobieren aufzuschreiben und zu begründen. Dieses Vorgehen dient zum einen dazu, dass die Fehlersuche zielgerichtet bleibt und nicht in reines Ausprobieren mündet. Zum anderen ermöglicht es den Schüler*innen, die für Naturwissenschaften und Technik übliche Abfolge von Vermuten, Überprüfen und Bestätigen / Widerlegen am Beispiel des Computational Thinking zu verinnerlichen.
Das Erarbeiten von Begründungen für die vermuteten Fehlerquellen und Lösungen ist darüber hinaus wichtig, um ein vertieftes Nachdenken über die Funktionsweise einzelner Befehle anzuregen. Dies ist eine Voraussetzung für das Generalisieren.

Für die Durchführung der Sequenz 3
verwenden Sie bitte die Karteikarten „Sequenz 3".

SEQUENZ 4
Abstrahieren anhand Calliope mini®

Kurzbeschreibung - Zentraler Inhalt:

Die Schüler*innen sollen in dieser Sequenz ein „automatisches Fahrradrücklicht" selbst *programmieren*. Ihnen wird als „Muster" ein echtes Rücklicht gezeigt, welches sich bei Abdunkelung einschaltet und bei normaler Tageshelligkeit wieder ausschaltet.

- *Zuerst* sollen die Schüler*innen das *Ziel* in eigenen Worten formulieren und *danach* überlegen, welche *Bausteine* sie zur Umsetzung benötigen. Hierbei kann wiederholt werden, welche Funktionen einzelne *Bausteine* haben und dass einzelne *Befehle* zu Abfolgen kombiniert werden können.
- Nun sollen die ausgewählten Bausteine in eine Abfolge gelegt werden. Dafür muss bedacht werden, was in welcher Reihenfolge überprüft und angezeigt werden soll, d.h. die grundlegende Struktur des Problems wird durchdacht und in Programmcode übersetzt.
- Das so entstandene Programm wird *überprüft*, eventuelle Fehler werden behoben.
- Schließlich werden weitere Lösungsmöglichkeiten erarbeitet und die Programmierung weiter optimiert. Ziel ist es, ein möglichst schlankes Programm zur Lösung der Problemstellung „automatisches Fahrradrücklicht" zu entwickeln.

Hintergrundinformationen und Hinweise

Wie auch in den vorherigen Sequenzen ist das Durchdenken der Aufgabenstellung und Planen der Lösungsschritte in dieser Sequenz zentral. Deshalb werden Lösungsentwürfe zunächst auf Papier gezeichnet oder mit Kärtchen gelegt, bevor die Programmierung am Calliope mini® ausprobiert wird. Mit Unterstützung der Lehrkraft können bereits bei der Programmplanung erste Fehler entdeckt und behoben werden, indem die Schüler*innen an Erkenntnisse aus den vorherigen Sequenzen erinnert werden. Die Schüler*innen sollten in dieser Sequenz außerdem gezielt dabei unterstützt werden (vgl. Kapitel 5), von der konkreten Fahrradlampe auf den grundlegenden Algorithmus zu schließen. Die für die Programmierung relevanten Merkmale der echten Fahrradlampe müssen von den nicht relevanten unterschieden und anschließend in die „Sprache" des Calliope mini® übersetzt werden.

Für die Durchführung der Sequenz 4
verwenden Sie bitte die Karteikarten „Sequenz 4".

SEQUENZ 5
Muster generalisieren anhand Calliope mini®

Kurzbeschreibung - Zentraler Inhalt:

Den Schüler*innen werden verschiedene Bausteine gezeigt, die das gleiche bewirken. Beispielsweise gibt es verschiedene Arten, den Buchstaben H auf dem LED-Display des Calliope mini® anzuzeigen.

- Um das Erkennen von Mustern und Regelmäßigkeiten anzuregen, sollen die Schüler*innen die vorgegebenen Bausteine anschauen, *vergleichen*, *Gemeinsamkeiten* und *Unterschiede* formulieren und überlegen was die Bausteine jeweils bewirken könnten.
- Ihre Erkenntnisse sollen sie in allgemeinen Regeln *festhalten*, z.B. in Form von *wenn-dann*-Sätzen.
- Zum *Überprüfen* der Regeln erfinden die Schüler*innen weitere Anwendungsbeispiele für ihre Regeln und beobachten anschließend am Calliope mini®, ob ihre Regel stimmt.

Das Erkennen und Formulieren von Regelmäßigkeiten ermöglicht es, verschiedene Programmierprobleme auf ihre zugrundeliegenden gemeinsamen Muster zurückzuführen.

Hintergrundinformationen und Hinweise

Beim Überprüfen der Regeln an von den Schüler*innen selbst gewählten neuen Programmierbeispielen muss beachtet werden, dass Unstimmigkeiten nicht nur auf falschen Regeln beruhen können, sondern auch daher rühren können, dass das gewählte Beispiel nicht zur Regel passt. Hier besteht eine Verbindung zum Thema „Modellieren" im Sachunterricht (vgl. z.B. Gogolin et al., 2017; Lange-Schubert & Hartinger, 2016): Modelle dienen in der Wissenschaft u.a. dazu, eine Hypothese zu überprüfen. Wird die Hypothese nicht bestätigt, muss überprüft werden, ob die Hypothese oder das Modell fehlerhaft waren.

Für die Durchführung der Sequenz 5
verwenden Sie bitte die Karteikarten „Sequenz 5".

SEQUENZ 6
Probleme und Prozesse zerlegen anhand Calliope mini®

Kurzbeschreibung - Zentraler Inhalt:

Die Aufgabe „Fahrradlicht mit Blink-Funktion" wird vorgestellt. Das Fahrradlicht soll so programmiert werden, dass gewählt werden kann, ob es dauerhaft leuchtet oder blinkt.

- Die Schüler*innen sollen nun überlegen, aus welchen *Teilzielen* die Problemstellung besteht und zunächst Lösungen für die Teilziele *planen*, programmieren und *evaluieren*, bis für jedes Teilproblem eine Lösung gefunden wurde. Da die Aufgabenstellung sehr offen ist, gibt es nicht nur verschiedene Lösungen für die Beleuchtung (Wahl der LED, ggf. der Farbe), sondern auch verschiedene Möglichkeiten, die Wahlmöglichkeit zu realisieren (z.B. per Knopfdruck, per Lagesensor).
- Wurden die Teilprobleme gelöst, müssen die Programmbausteine in ein Gesamtprogramm zur Gesamtlösung zusammengesetzt werden. Auch hier wird zunächst die Lösung geplant und anschließend programmiert und evaluiert.
- Zeigen sich beim Gesamtprogramm Abweichungen von der angestrebten Funktion, kann die Ursache entweder in der Reihenfolge der Programmbausteine liegen oder es müssen beim Zusammenfügen noch weitere Bausteine integriert werden (bspw. weitere Logik-Bausteine, wenn im laufenden Betrieb zwischen den Funktionen gewechselt werden soll).

Hintergrundinformationen und Hinweise

Die vorgeschlagene Aufgabenstellung kann auf unterschiedlichen Komplexitätsniveaus realisiert werden (Differenzierung). Die höchste Komplexität weist ein Lampenprogramm auf, das mehrfach im laufenden Betrieb zwischen Dauerleuchten, Blinken und Nichtleuchten umgeschaltet werden kann.
Einfacher zu programmieren ist eine Lampe, die nach Start des Calliope mini® nur eine einzelne Eingabe (z.B. Knopf A für Dauerleuchten) akzeptiert und dann bis zum Ausschalten bzw. Resetten des Calliope mini® in diesem Zustand (z.B. Dauerleuchten) verbleibt. Möchte man z.B. auf Blinken umschalten, muss der Calliope mini® bei dieser Programmierung zunächst neu gestartet werden, da das Programm sonst keine neue Eingabe berücksichtigt.
Das Zusammenfügen der Programmbausteine kann den Schüler*innen als „Knobelaufgabe" vorgestellt werden

Für die Durchführung der Sequenz 6
verwenden Sie bitte die Karteikarten „Sequenz 6".

5. Adaptive Lernbegleitung

In diesem Kapitel erhalten Sie vertieftes Wissen dazu, wie Sie Unterstützungsmaßnahmen zum Computational Thinking gezielt planen und in Ihren Unterricht einbinden können, um den fachlichen und sprachlichen Kompetenzaufbau der Schüler*innen mittels problemorientierter Aufgaben zu fördern. Problemorientierte Aufgaben ermöglichen einerseits, die aktuellen Kompetenzen des Kindes zu erfassen und bieten andererseits den Rahmen für gezielte Unterstützungsmaßnahmen zur Kompetenzerweiterung. Deshalb sind problemorientierte Aufgaben besonders geeignet für das Formative Assessment und daran anknüpfendes Scaffolding.

Formative Assessment

Beim Formative Assessment werden aktuelle Lernvoraussetzungen und die Lernentwicklung der Schüler*innen während des Lehr-Lern-Prozesses mehrfach diagnostiziert, um das Unterrichtsangebot fortwährend an die Lernentwicklung anzupassen und den Schüler*innen ihren Lernstand rückzumelden. Hierfür können gezielt Aufgaben als Leistungstest eingesetzt werden oder es können informell Rückschlüsse aus dem Unterrichtsgeschehen gezogen werden, damit darauf aufbauend passgenaue Unterstützungsmaßnahmen geplant und eingesetzt werden können (Krause & Stark, 2006; Shavelson, 2006).
Die problemorientierten Aufgaben in den Sequenzen (vgl. Kapitel 4 und Karteikarten) nehmen jeweils den Aufbau zu einzelnen Kompetenzen des Computational Thinking besonders in den Blick. Durch das Beobachten und Unterstützen der Schüler*innen beim Bearbeiten der Aufgaben (z.B. in Kleingruppen) und durch das Analysieren ihrer Programmierlösungen (oder Zwischenlösungen) können Informationen darüber gewonnen werden, welche Kompetenzaspekte von Schüler*innen bereits beherrscht werden und wo noch Schwierigkeiten bestehen. Die Impulse der Karteikarten schlagen konkrete Maßnahmen vor, mit denen Einblicke in die aktuellen Computational-Thinking-Kompetenzen der Schüler*innen gewonnen werden können. Bspw. kann beobachtet werden, nach welchen Kriterien die Befehle in Sequenz 1 geordnet werden, welche Vermutungen Schüler*innen über die Funktion von Befehlen oder Befehlsabfolgen äußern, welche Fehler bei den Aufgaben der einzelnen Sequenzen gemacht werden und welche Fehler bereits nicht mehr gemacht werden.
Ergänzend zu diesen eher informellen Unterrichtsbeobachtungen können auch Test- oder Selbsttestaufgaben für die Schüler*innen entwickelt werden, um sich selbst und den Schüler*innen einen systematischen Einblick in deren individuellen Lernstand zu ermöglichen. Auch beim Einsatz von Testaufgaben steht im Rahmen des formativen Assessment nicht die Vergabe einer Note im Vordergrund, sondern die Analyse des aktuellen Lernstandes und Lernverlaufes, um in den darauffolgenden Lehr-Lern-Situationen zum Computational Thinking daran anzuknüpfen.

Um auch das sprachliche Lernen der Schüler*innen zu unterstützen, sollten zusätzlich Beobachtungen zu den von den Schüler*innen verwendeten (bzw. noch nicht oder inkorrekt verwendeten) Begriffen und Formulierungen gemacht werden. Auch hier können ggf. systematische Verfahren (z.B. Wortschatztests oder das schriftliche Verfassen von Vermutungen und Begründungen in Einzelarbeit) eingesetzt werden, um bei allen Schüler*innen einen Einblick in die themenbezogenen sprachlichen Kompetenzen zu erhalten.

Für gelingendes Formative Assessment ist es zudem wichtig, dass die Lehrkraft ein kollaboratives, fehlertolerantes Arbeitsklima in der Klasse etabliert und bei den Schüler*innen das Bewusstsein schafft, dass sie selbst (mit)verantwortlich für ihren Lernprozess sind (Liebers & Seifert, 2012).
Die im Kontext des Formative Assessment geleisteten Unterstützungsmaßnahmen werden im Sachunterricht unter dem Begriff Scaffolding zusammengefasst.

Scaffolding

Scaffolding bedeutet, dass Lernende genau die Unterstützung erhalten, die sie benötigen, um CT-Probleme lösen zu können, die sie ohne Unterstützung noch nicht lösen könnten. Im Verlauf des Lernprozesses sollen sie die zur Aufgabenlösung notwendigen Denk-, Arbeits- und Handlungsweisen mehr und mehr selbst übernehmen und so schrittweise die Kompetenzen zur Lösung immer komplexerer Programmierprobleme erwerben (vgl. auch van de Pol et al., 2010).
Scaffolding ist nicht vom Formative Assessment zu trennen, da adäquate Scaffolding-Maßnahmen auf vorheriger Diagnose basieren. Scaffolding-Maßnahmen beinhalten wiederum implizite oder explizite Rückmeldungen an die Schüler*innen (Bürgermeister & Saalbach, 2018).
Scaffolding kann in drei Arten untergliedert werden (Shavelson, 2006):

(1) *Scaffolding embedded in the curriculum*, in welchem Aufgaben so konzipiert sind, dass sie die Schüler*innen beim Lernen unterstützen.
(2) *Scaffolding planned for interaction*, mit dem Unterstützungsmaßnahmen für den Einsatz während des Unterrichts geplant werden.
(3) *on-the-fly-Scaffolding*, also den adaptiven Einsatz der Unterstützungsmaßnahmen im Unterricht.

Scaffolding embedded in the curriculum

Scaffolding embedded in the curriculum schafft die Rahmenbedingungen für scaffolding planned for interaction und on-the-fly-Scaffolding. Bspw. werden problemorientierte Aufgaben in das Bildungsangebot integriert oder in jeder Sequenz andere Schwerpunkte hinsichtlich der zu erlernenden CT-Kompetenzen, Begrifflichkeiten und Sprachhandlungen gesetzt. Auf den Karteikarten sind bereits einige Anregungen zum Unterstützen des

Problemlöseprozesses enthalten. Weiterhin kann es aufgrund der komplexen Denkprozesse beim Computational Thinking sinnvoll sein, in der Unterrichtssituation das kognitive Lösen des Problems von der Anforderung an sprachlich adäquate Formulierungen zu trennen (vgl. Gibbons, 2006). Während der Problemlösephase in den Kleingruppen tritt die sprachliche Formulierung in den Hintergrund. Die Schüler*innen können sich in dieser Arbeitsphase ggf. auch durch Zeigen auf Befehlskarten oder Bauteile des Calliope mini® untereinander oder mit der Lehrkraft verständigen. Erst beim Formulieren des Gruppenberichts in den Kleingruppen (nach erfolgter Problemlösung) und beim Vorstellen der Ergebnisse im Plenum rückt die sprachliche Ausdrucksweise und somit auch die Sprachförderung in den Fokus.

Das Erarbeiten eines Wort- und Satzspeichers mit wichtigen Begriffen und Formulierungen ist ebenfalls eine sinnvolle Maßnahme des *Scaffolding embedded in the curriculum*, um einerseits den Austausch über Computational Thinking und andererseits das fachsprachliche Lernen zu unterstützen. Möglicherweise bedeutsame Begriffe für den Wortspeicher zu den Computational-Thinking-Sequenzen sind in Kapitel 4 und auf den Karteikarten bereits hervorgehoben. Ergänzend können z.B. Glossare in Lehrmaterialien (vgl. z.B. Abend et al., 2017; Körner & Bettner, 2021) Hinweise auf bedeutsamen Wortschatz für die jeweilige Sequenz und Lerngruppe geben. Für einen Satzspeicher können themenspezifische Hilfestellungen (z.B. für Debugging: „Wir haben den Fehler behoben, indem …", für Wirkungen: „Baustein … bewirkt, dass …") mit Satzanfängen für bedeutsame Sprachhandlungen bzgl. der Denk-, Arbeits- und Handlungsweisen wie Vermuten („Ich vermute, dass …") oder Begründen („… ist passiert, weil …") erweitert werden. Der Wort- und Satzspeicher sollte so übersichtlich gestaltet sein, dass die benötigten Worte und Formulierungshilfen mit einem Blick gefunden werden können (DZLM & PIKAS, 2022). Er kann z.B. als Plakat oder einzelne Karten im Raum aufgehängt werden, an der Tafel entwickelt werden oder in Form einer Mindmap gestaltet werden.
Beim Erarbeiten der benötigten sprachlichen Strukturen kann z.B. ein *Planungsrahmen* zur Sprachförderung eingesetzt werden (vgl. Quehl & Trapp, 2013). Er hilft dabei, die sprachlichen Anforderungen zum Thema „Computational Thinking mit Calliope mini®" hinsichtlich der für die Unterrichtsaktivitäten notwendigen Sprachhandlungen und Satzstrukturen sowie des Vokabulars zu analysieren.

Mit Unterstützungsmaßnahmen, die bereits im Vorfeld in das Unterrichtsangebot integriert werden können, wird der Lehr-Lern-Prozess vorstrukturiert (Shavelson, 2006). Sie sind bereits Teil des hier vorgestellten Bildungsangebots, müssen aber noch an die Besonderheiten der jeweiligen Schulklasse angepasst und in die Unterrichtsinteraktion eingebunden werden (vgl. *Scaffolding planned for interaction*).

Scaffolding planned for interaction

Scaffolding planned for interaction schafft die Rahmenbedingungen für *on-the-fly-scaffolding*, indem zu jeder Aufgabe im Vorfeld genau überlegt wird, welchen möglichen Herausforderungen die Schüler*innen beim Bearbeiten eines CT-Problems begegnen werden.

Sodann wird eine Vielfalt an möglichen Scaffolding-Maßnahmen wie z. B. Hilfekarten, Fragen oder Hinweisen geplant, um diese bei Bedarf einsetzen zu können. Die Impulse auf den Karteikarten bieten erste Anhaltspunkte für Fragen und Hinweise, die zur Unterstützung der Schüler*innen beim Problemlösen eingesetzt werden können. Diese können und sollten um weitere konkret formulierte Unterstützungsmaßnahmen für die jeweilige Aufgabe und Schülergruppe erweitert werden.
Auch die Anpassung des Wort- und Satzspeichers an die jeweilige Lerngruppe und Überlegungen dazu, wie er im Unterricht eingebunden werden kann, gehören zum *Scaffolding planned for interaction*. Im Calliope-Bildungsangebot kann die Lehrkraft wichtige Worte und Formulierungen z.B. in den Gruppenberichten der Kleingruppen beobachten und anschließend bei der Reflexion gezielt aufgreifen (vgl. Aufbau der Doppelstunde in Kapitel 2). Kommen in den Berichten der Schüler*innen schon zentrale fachsprachliche Ausdrücke vor, kann die Lehrkraft diese noch einmal vorlesen lassen und sie dann in den Wort- und Satzspeicher aufnehmen. Ergänzend kann die Lehrkraft selbst weitere mit Hilfe des Planungsrahmens vorbereitete Worte und Formulierungen einbringen, um den Wort- und Satzspeicher zu vervollständigen (DZLM & PIKAS, 2022).
Um Schüler*innen bei der Erweiterung ihrer sprachlichen Kompetenzen zu unterstützen sind darüber hinaus sprachförderliche Interaktionen bedeutsam. Hierfür kann die Lehrkraft bspw. eine Schüleräußerung in erweiterter oder korrigierter Form wiederholen, selbst den erwarteten Sprachgebrauch demonstrieren oder durch das Stellen offener Fragen Schüler*innen zu umfangreicheren Antworten herausfordern (vgl. z.B. Kammermeyer et al., 2017; Wildemann & Merkert, 2020).

On-The-Fly-Scaffolding

Die tatsächlich im Unterricht vorgenommene flexible und adaptive, passgenaue Unterstützung aufgrund einer situativen Diagnose wird als *on-the-fly-scaffolding* bezeichnet. Es kann nur gelingen, wenn zuvor *Scaffolding embedded-in-the-curriculum* und *scaffolding-planned-for-interaction* möglichst genau und umfassend vorgenommen wurden.
Neben dem Einsatz konkreter Unterstützungsmaßnahmen ist es während der Unterrichts-interaktion zentral, den Schüler*innen ausreichend Zeit zum Nachdenken über Programmierprobleme und mögliche Lösungen sowie zur sprachlichen Formulierung ihrer Antworten zur Verfügung zu stellen.
Der Wort- und Satzspeicher enthält ebenfalls Aspekte des on-the-fly-Scaffolding: Er wird gemeinsam mit den Schüler*innen im Unterricht erarbeitet und die Lehrkraft weist die Schüler*innen bei Formulierungsaufgaben (z.B. folgende Gruppenberichte) oder sprachlichen Schwierigkeiten auf den Wort- und Satzspeicher hin. Dies ist wichtig, damit die Schüler*innen ihn als hilfreiche Ressource wahrnehmen, um alltagssprachliche Ausdrücke in bildungs- und fachsprachliche Formulierungen zu überführen (DZLM & PIKAS, 2022).

6. Konkrete Planungshilfen

Die in Kapitel 5 genannten Hilfen zur Planung der Scaffolding-Maßnahmen werden im Folgenden an den Inhalten der Sequenz 3 zur Kompetenz Debugging illustriert, um zu verdeutlichen, wie diese Planungshilfen beim Lehr-Lern-Angebot zum Computational Thinking genutzt werden können.

Zunächst wird eine Planungshilfe für das scaffolding planned for interaction bzgl. des Problemlösens in Sequenz 3 vorgestellt. Die Tabelle enthält einige mögliche Schwierigkeiten, die Schüler*innen beim Bearbeiten der CT-Aufgabe von Sequenz 3 haben könnten und zeigt dazu passende Unterstützungsmaßnahmen und deren Zielsetzung. Darauf folgen die Planungshilfen für das Scaffolding von Sprache, welches wichtig ist, um die Schüler*innen auch beim Erfüllen der kommunikativen Anteile des Bildungsangebotes unterstützen zu können. Dies ist z.B. auch dann wichtig, wenn die Lehrkraft beim Scaffolding des Problemlösens Warum-Fragen stellt (vgl. Planungshilfe für das Scaffolding planned for interaction bzgl. des Problemlösens): Wenn Schüler*innen beim Beantworten dieser Fragen Schwierigkeiten haben, ihre Denkprozesse in Worte zu fassen, kann das Scaffolding von Sprache auch elementar für das Scaffolding des Problemlöseprozesses werden. Die Planungshilfen für das Scaffolding von Sprache zeigen zuerst den Planungsrahmen Sprache. Der Planungsrahmen gehört zum Scaffolding embedded in the curriculum und hilft der Lehrkraft dabei, sich die konkreten sprachlichen Anforderungen bewusst zu machen, die mit der Bearbeitung des jeweiligen CT-Problems verbunden sind. Das CT-Problem spiegelt sich in den Spalten „Thema" und „Aktivitäten" des Planungsrahmens wider. Die Aktivitäten werden zunächst in Sprachhandlungen (z.B. Beschreiben, Vermuten, Begründen) übersetzt, welche eng mit naturwissenschaftlich-technischen Denk-, Arbeits- und Handlungsweisen verknüpft und auch für das Sprechen über CT-Probleme und deren Lösungen unerlässlich sind. Anschließend wird analysiert, welche Satzstrukturen und welcher Wortschatz dafür konkret benötigt werden. Dies ist nicht nur vom jeweiligen Fachinhalt (Thema, Aktivitäten) abhängig, sondern auch von den sprachlichen Fähigkeiten der Lernenden.

Aus dem Planungsrahmen kann der Inhalt des Wort- und Satzspeichers für die jeweilige Sequenz abgeleitet werden. Dieser gehört ebenfalls zum Scaffolding embedded in the curriculum.

Schließlich werden mögliche verbale Unterstützungsmaßnahmen zum Scaffolding von Sprache in Sequenz 3 vorgestellt. Sie gehören zum scaffolding planned for interaction und sind analog zur ersten Planungshilfe dieses Kapitels (scaffolding planned for interaction bzgl. des Problemlösens) aufgebaut. Es werden zunächst mögliche sprachliche Schwierigkeiten / Herausforderungen der Sequenz für Schüler*innen aufgezeigt und anschließend dazu passende Unterstützungsmaßnahmen formuliert und benannt.

Die dargestellten Planungshilfen müssen an die jeweilige Sequenz und an die Lerngruppe, mit der das Bildungsangebot durchgeführt wird, angepasst werden. Daher ist auch eine Überprüfung und ggf. Erweiterung oder Veränderung der Planungshilfen für Sequenz 3 notwendig.

Planungshilfe Scaffolding planned for Interaction: Computational Thinking

Die in der Planungshilfe aufgeführten Beispiele orientieren sich an den Inhalten der Sequenz 3 „Debugging“.

Mögliche Schwierigkeiten / Herausforderungen	**Mögliche passende Unterstützungsmaßnahmen**	**Ziel der Maßnahme**
Das Programm wird nur im Ganzen betrachtet	Zerlege das Programm in Bausteine und überlege, was die einzelnen Bausteine bewirken.	einen Hinweis geben, der die komplexe Aufgabe in handhabbare Teilschritte zerlegt.
Bei der Analyse der Einzelbausteine wird der Fehler nicht gefunden oder bei der schwierigen Variante werden nicht alle Fehler gefunden	Überlege wo der Fehler liegen könnte: - Ist ein überflüssiger Baustein dabei? - Wurde etwas vergessen? - Vielleicht gibt es auch mehrere Fehler. - Wie würdest du (z.B.) die Hupe programmieren?	Die erste Frage regt das Analysieren der vorhanden Bausteine an, die übrigen Fragen gehen darüber hinaus.
Es wird herumprobiert, ohne nachzudenken	Erkläre mir mal, was du gerade machst. Warum denkst du, dass der Fehler an dieser Stelle / diesem Baustein liegt? Warum entfernst du diesen Baustein? [Hilfsfrage: Was macht der Baustein?] Warum fügst du diesen Baustein ein? [Hilfsfrage: Was macht der Baustein?] Warum an dieser Stelle?	Einfordern von Erklärungen bzw. Begründungen , um zum Analysieren der Bausteine und Verstehen des Programmcodes sowie zum geplanten Fehlersuchen anzuregen.
…		

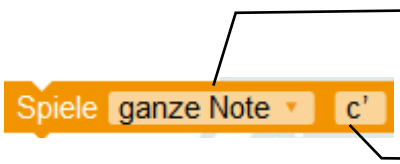
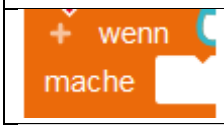
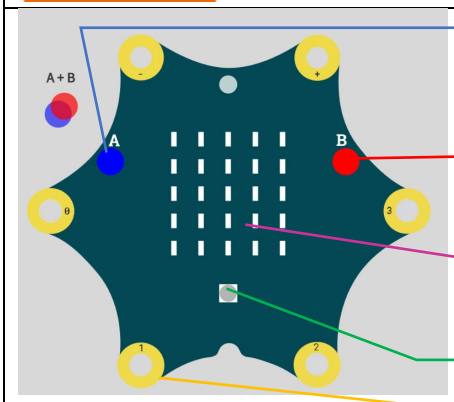
Planungsrahmen Sprache beim Computational Thinking am Bsp. Debugging

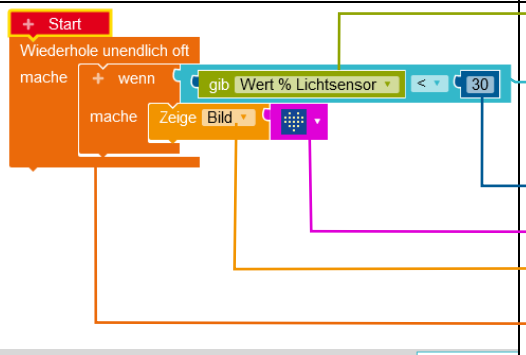
Thema	Aktivitäten	Sprachhandlungen	Satzstrukturen	Wortschatz
Computational Thinking: Fokus auf Debugging	Fehler erkennen und beschreiben	**beschreiben**, was das Programm bewirken soll und was es tatsächlich bewirkt. Abweichung von Ist- und Soll-Funktionsweise formulieren.	*Wenn* ich A drücke, [*dann*] passiert nichts, *obwohl* Calliope mini® jetzt einen Ton abspielen *sollte*.	Programm Fehler abspielen ...
	Fehlerort und Art des Fehlers identifizieren	**Benennen** welche Bausteine verwendet wurden. **Beschreiben**, was einzelne Bausteine bewirken. **Vermuten,** wo der Fehler liegt / wodurch der Fehler verursacht wird	Das Programm besteht aus dem Kontrollbaustein „wenn + mache", dem Sensorbaustein „Taste A gedrückt" und dem Aktionsbaustein „Spiele ganze Note c". Das Programm lässt Calliope mini® einen Ton abspielen, wenn Taste A gedrückt wird. *Ich vermute, dass* der Fehler darin liegt, dass ... das Programm nur einmal ausgeführt wird. Es ist vielleicht schon fertig gelaufen, *bevor* wir A drücken *und deshalb* wird kein Ton mehr abgespielt, *obwohl* A gedrückt ist.	Programm Kontrollbaustein Sensorbaustein Aktionsbaustein Fehler besteht aus abspielen ausführen ...

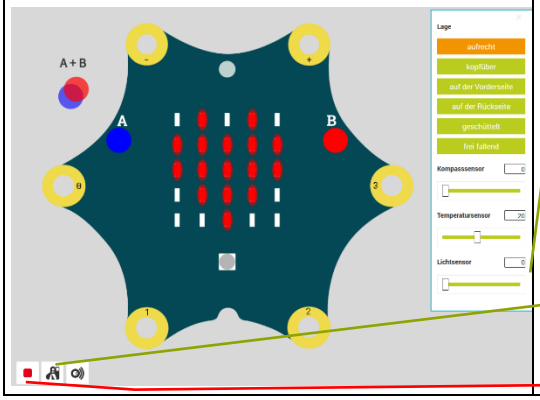
	Fehler beheben	Planen und **Begründen** der Lösung Überprüfen der Lösungsidee und **Beschreiben** des Effekts.	Ich baue den Kontrollbaustein „Wiederhole unendlich oft“ so ein, dass er die anderen Bausteine des Programmcodes umschließt, weil Calliope mini® dann immer wieder überprüft, ob A gedrückt wurde. Dann müsste er auch den Ton abspielen, wenn man A drückt. Jetzt führe ich das Programm aus und kann hören, dass Calliope mini® den Ton abspielt, wenn ich A drücke. Das Problem ist also gelöst.	Kontrollbaustein Programm Problem abspielen ausführen lösen/gelöst …

Wort- und Satzspeicher

Ein Wort- und Satzspeicher, der im Laufe der Sequenzen erweitert werden kann, könnte u.a. Folgendes enthalten:

<table>
<tr><th>Baustein</th><th>Wort- und Satzspeicher</th></tr>
<tr><td> 
</td><td>Notenlänge
Tonhöhe
Calliope mini® spielt
eine halbe Note der Tonhöhe c'.</td></tr>
<tr><td> 
</td><td>Wenn die Bedingung … erfüllt ist,
führt Calliope mini® den Befehl … aus.</td></tr>
<tr><td> 

Tipp:[4]</td><td>Taste A
Taste B
LED-Display
RGB-LED
Pin 1</td></tr>
<tr><td> 

 
</td><td>Sensorbaustein
Logikbaustein
Mathematikbaustein
Bilderbaustein
Aktionsbaustein
Kontrollbaustein
Eine Befehlsabfolge, ein Programm
Das Programm besteht aus…
Der Lichtsensor hat den Wert 0.
Anzeige der Sensordaten öffnen
Programm ausführen / Ausführung stoppen</td></tr>
</table>

[4] Beschriftete Abbildungen, die die Bauteile des Calliope mini® zeigen, finden sich auch in anderen Veröffentlichungen zum Calliope mini® (vgl. z.B. Abend et al. (2017); Bockermann et al. (2018)).

Planungshilfe Scaffolding planned for Interaction: Bildungssprache

Die in der Planungshilfe aufgeführten Beispiele orientieren sich an den Inhalten der Sequenz 3 „Debugging".

Mögliche Schwierigkeiten / Herausforderungen	**Mögliche passende Unterstützungsmaßnahmen**	**Name der Maßnahme**
Schüler*innen verwenden alltagssprachliche oder erfundene Begriffe, z.B. „Calliope mini® macht möp!"	Rückfragen stellen oder auf Wortspeicher verweisen und um Reformulierung bitten, z.B. Schau mal im Wortspeicher nach, wie du das noch genauer sagen kannst.	 Verweis auf Wortspeicher
„Wenn ich den Knopf drücke, passiert ja gar nichts!"	Welchen Knopf drückst du denn? [Schüler*in zeigt auf Knopf A: „Das da."] Wie heißt denn dieser Knopf am Calliope mini®? Der Wortspeicher kann dir dabei helfen. [Schüler*in liest ab: Taste A] auch möglich wäre: Drückst du Taste A oder Taste B?	Offene Frage Offene Frage + Verweis auf Wortspeicher Alternativfrage, die benötigten Wortschatz enthält
Schüler*innen haben Schwierigkeiten mit der Grammatik, z.B. „Ich vermute, das Fehler ist, die Programm läuft nur einmal."	Das Gesagte in korrigierter Form aufgreifen und die korrigierte Stelle betonen, z.B. Du vermutest, **dass der** Fehler ist, **dass das** Programm nur einmal läuft.	 Indirekte Korrektur

Mögliche Schwierigkeiten / Herausforderungen	**Mögliche passende Unterstützungsmaßnahmen**	**Name der Maßnahme**
Schüler*innen beschreiben Programmabläufe ungenau / unvollständig, z.B.: Calliope mini® startet erstmal mit warten. Dann spielt er eine halbe Note. 	Rückfragen stellen, ggf. auf Wort- und Satzspeicher verweisen oder Sprachmodell sein, z.B. Wie lange wartet Calliope mini® mit dem Spielen der Note? [Schüler*in sagt: „Bis ich Taste A drücke."] Aha! Calliope mini® wartet, bis Taste A gedrückt wird. Wie kannst du das in deine Beschreibung einfügen? (ggf.: Im Wortspeicher findest du Satzbausteine, die dir helfen.) Und welche Tonhöhe hat die Note?	 Offene Frage Umformulierung Offene Frage (Verweis auf Wortspeicher) Quizfrage, die benötigten Wortschatz enthält.
...		

Literaturverzeichnis

Abend, M., Gramowski, K., Pelz, L. & Poloczek, B. (2017). *Coden mit dem Calliope mini: Programmieren in der Grundschule*. Lehrmittel für den Einsatz ab Klasse 3 (1. Auflage). *Für Lehrer*. Cornelsen.

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J. & Zagami, J. (2016). A K-6 Computational Thinking Curriculum Framework: Implications for Teacher Knowledge. *Educational Technology & Society*, *19*(3), 47–57.

Bergner, N., Franken, P., Kleeberger, J., Leonhardt, T., Lukas, M., Pesch, M., Prost, N., Thar, J. & Wassong, L. (2017). *Das Calliope-Buch: Spannende Bastelprojekte mit dem Calliope-mini-Board* (1. Auflage, 1. korrigierter Nachdruck). dpunkt.verlag.

Bergner, N. & Leonhardt, T. (2018). *Programmieren mit dem Calliope mini: für Dummies junior*. Wiley.

Bockermann, I., Engelbertz, S., Illginnis, S., Moebus, A., Murmann, L., Reid, D. A. & Schelhowe, H. (2018). *Lehrerhandreichung zur Bremer Explorationsstudie Calliope mini: Modul 1*. Was ist Programmieren? https://media.suub.uni-bremen.de/bitstream/elib/3502/1/00107005-1.pdf

Böschl, F., Gogolin, S., Lange-Schubert, K. & Hartinger, A. (2018). Modellverstehen von Grundschüler/innen in Abhängigkeit von Kontext und Kompetenznive. In U. Franz, H. Giest, A. Hartinger, A. Heinrich-Dönges & B. Reinhoffer (Hrsg.), *Probleme und Perspektiven des Sachunterrichts: Band 28. Handeln im Sachunterricht* (S. 93–100). Verlag Julius Klinkhardt.

Bürgermeister, A. & Saalbach, H. (2018). Formatives Assessment: Ein Ansatz zur Förderung individueller Lernprozesse. *Psychologie in Erziehung und Unterricht*, *65*(3), 194–205. https://doi.org/10.2378/peu2018.art11d

Criblez, L., Oelkers, J., Reusser, K., Berner, E., Halbheer, U. & Huber, C. (2009). *Bildungsstandards* (1. Aufl.). *Lehren lernen - Basiswissen für die Lehrerinnen- und Lehrerbildung*. Kallmeyer/Klett, Klett und Balmer. https://doi.org/8013

Crismond, D. (2001). Learning and using science ideas when doing investigate-and-redesign tasks: A study of naive, novice, and expert designers doing constrained and scaffolded design work. *Journal of Research in Science Teaching*, *38*(7), 791–820. https://doi.org/10.1002/tea.1032

Curzon, P., Dorling, M., Ng, T., Selby, C. & Woollard, J. (2014). *Developing computational thinking in the classroom: a framework*. Computing At School. https://eprints.soton.ac.uk/369594/1/DevelopingComputationalThinkingInTheClassroomaFramework.pdf

Di Lieto, M. C., Inguaggiato, E., Castro, E., Cecchi, F., Cioni, G., Dell'Omo, M., Laschi, C., Pecini, C., Santerini, G., Sgandurra, G. & Dario, P. (2017). Educational Robotics intervention on Executive Functions in preschool children: A pilot study. *Computers in Human Behavior*, *71*, 16–23. https://doi.org/10.1016/j.chb.2017.01.018

Dickes, A. C., SENGUPTA, P., Farris, A. V. & BASU, S. (2016). Development of Mechanistic Reasoning and Multilevel Explanations of Ecology in Third Grade Using Agent-Based Models. *Science Education*, *100*(4), 734–776. https://doi.org/10.1002/sce.21217

DZLM & PIKAS (Hrsg.). (2022). *Unterricht: Mathematikunterricht sprachsensibel gestalten*. https://primakom.dzlm.de/node/223

Ebel, M. (2020). *Förderung von naturwissenschaftlichen Konzepten und Bildungssprache von Vorschulkindern: Effekte kontext-reduzierter Gespräche auf konzeptuelle Vorstellungen zu Hebelwirkung und bildungssprachliche Lexik und Grammatik* [Dissertation]. Universität Koblenz-Landau, Landau. https://d-nb.info/1205315497/34

Faber, H. H., Wierdsma, M. D. M., Doornbos, R. P., van der Ven, J. S. & Vette, K. de (2017). Teaching computational thinking to primary school students via unplugged programming lessons. *Journal of the European Teacher Education Network*, *12*, 13–24. https://drive.google.com/file/d/1c93Xk3L11nqrKnV5SnxUvPY4QZJthUaC/view

Feierabend, S., Rathgeb, T., and Reutter, T. (2019). *KIM-Studie 2018: Kindheit, Internet, Medien*. Basisuntersuchung zum Medienumgang 6-bis 13-Jähriger. https://www.mpfs.de/fileadmin/files/Studien/KIM/2018/KIM-Studie_2018_web.pdf

Funke, J. (2003). *Problemlösendes Denken* (1. Auflage). *Kohlhammer Standards Psychologie*. Verlag W. Kohlhammer.

Furtak, E. M., Seidel, T., Iverson, H. & Briggs, D. C. (2012). Experimental and Quasi-Experimental Studies of Inquiry-Based Science Teaching. *Review of Educational Research*, *82*(3), 300–329. https://doi.org/10.3102/0034654312457206

Gibbons, P. (2006). *Bridging discourses in the ESL classroom: Students, teachers and researchers*. Continuum.

Gogolin, S., Krell, M., Lange-Schubert, K., Hartinger, A., Upmeier zu Belzen, A. & Krüger, D. (2017). Erfassung von Modellkompetenz bei Grundschüler/innen. In H. Giest, A. Hartinger & S. Tänzer (Hrsg.), *Probleme und Perspektiven des Sachunterrichts: Band 27. Vielperspektivität im Sachunterricht* (S. 108–115). Verlag Julius Klinkhardt.

Grygier, P. & Hartinger, A. (2012). *Gute Aufgaben Sachunterricht: Naturwissenschaftliche Phänomene begreifen*. 48 gute Aufgaben für die Klassen 1 bis 4 (2. Auflage). *Lehrerbücherei Grundschule*. Cornelsen.

Hofer, S. I., Schumacher, R., Rubin, H. & Stern, E. (2018). Enhancing physics learning with cognitively activating instruction: A quasi-experimental classroom intervention study. *Journal of Educational Psychology*, *110*(8), 1175–1191. https://doi.org/10.1037/edu0000266

Kammermeyer, G., Goebel, P., King, S., Lämmerhiert, A., Leber, A., Metz, A., Papillon-Piller, A. & Roux, S. (2017). *Mit Kindern im Gespräch (Grundschule): Strategien zur Sprachbildung und Sprachförderung von Kindern in der Grundschule* (1. Auflage). *Grundschule*. Auer. https://doi.org/Gisela

Kiefer, P. (2018). *Calliope mini: Coden, basteln, entdecken* (1. Auflage). Vierfarben; Rheinwerk Verlag.

Körner, M. & Bettner, M. (2021). *Programmieren mit dem Calliope mini - Grundschule: Motivierende Aufgaben und kleine Projekte zum Tüfteln und Programmieren* (1. Auflage). *Bergedorfer Unterrichtsideen*. Persen.

Krause, U.-M. & Stark, R. (2006). Vorwissen aktivieren. In H. Mandl & H. F. Friedrich (Hrsg.), *Handbuch Lernstrategien* (S. 38–49). Hogrefe.

Lange-Schubert, K. & Hartinger, A. (2016). Wege zur Modellierungskompetenz von Grundschüler/innen im Sachunterricht. In T. Goll & A. Hartinger (Hrsg.), *Sachunterricht – zwischen Kompetenzorientierung, Persönlichkeitsentwicklung, Lebenswelt und Fachbezug* (S. 66–74). Verlag Julius Klinkhardt.

Lazonder, A. W. & Harmsen, R. (2016). Meta-Analysis of Inquiry-Based Learning. *Review of Educational Research*, *86*(3), 681–718. https://doi.org/10.3102/0034654315627366

Liebers, K. & Seifert, C. (2012). Assessmentkonzepte für die inklusive Schule – eine Bestandsaufnahme. *Zeitschrift für Inklusion*(3). http://www.inklusion-online.net/index.php/inklusion-online/article/view/44/44

Lucas, B., Hanson, J. & Claxton, G. (2014). *Thinking like an engineer: Implications for the education system*. Royal Academy of Engineering.

Paniagua, A. & Istance, D. (2018). *Teachers as Designers of Learning Environments: The Importance of Innovative Pedagogies*. Educational Research and Innovation,. OECD Publishing.

Quehl, T. & Trapp, U. (2013). *Sprachbildung im Sachunterricht der Grundschule: Mit dem Scaffolding-Konzept unterwegs zur Bildungssprache*. *FörMig Material: Bd. 4*. Waxmann.

Relkin, E. & Strawhacker, A. (2021). Unplugged Learning: Recognizing Computational Thinking in Everyday Life. In M. Bers (Hrsg.), *Advances in early childhood and K-12 education (AECKE) book series. Teaching computational thinking and coding to young children* (S. 41–62). IGI Global. https://doi.org/10.4018/978-1-7998-7308-2.ch003

Reusser, K. (2005). Problemorientiertes Lernen – Tiefenstruktur, Gestaltungsformen, Wirkung. *Beiträge zur Lehrerbildung*, *23*(2), 159–182.

Reuter, T. & Leuchter, M. (2019). Lernwirksamkeit unterschiedlich strukturierter Lern angebote zu Zahnrädern in der Grundschule. In M. Knörzer, L. Förster, A. Hartinger & U. Franz (Hrsg.), *Probleme und Perspektiven des Sachunterrichts: Band 29. Forschendes Lernen im Sachunterricht* (S. 55–62). Verlag Julius Klinkhardt.

Schmitt, L., Weber, A., Weber, D. & Leuchter, M. (2023). First Insights into Preschool Teachers' Instructional Quality in Block Play and Its Associations with Children's Knowledge, Interest, Academic Self-Concept and Cognitive Aspects. *Early Education and Development*, 1–23. https://doi.org/10.1080/10409289.2023.2233879

Selby, C. C. (2014). *How can the teaching of programming be used to enhance computational thinking skills?* [unveröffentlichte Dissertation]. University of Southampton, Southampton, UK.

Shavelson, R. J. (2006). On the Integration of Formative Assessment in Teaching and Learning: Implications for New Pathways in Teacher Education. In F. K. Oser, F. Achtenhagen &

U. Renold (Hrsg.), *Competence Oriented Teacher Training: Old Research Demands and New Pathways* (S. 61–78). BRILL. https://doi.org/10.1163/9789087903374_006

Shute, V. J., Sun, C. & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*(1), 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

van de Pol, J., Volman, M. & Beishuizen, J. (2010). Scaffolding in Teacher–Student Interaction: A Decade of Research. *Educational Psychology Review*, *22*(3), 271–296. https://doi.org/10.1007/s10648-010-9127-6

Vollmer, H. J. & Thürmann, E. (2013). Sprachbildung und Bildungssprache als Aufgabe aller Fächer der Regelschule. In M. Becker-Mrotzek, K. Schramm, E. Thürmann & H. J. Vollmer (Hrsg.), *Sprache im Fach: Sprachlichkeit und fachliches Lernen* (S. 41–57). Waxmann.

Vygotsky, L. S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press.

Weber, A., Barkela, V., Stiel-Dämmer, S. & Leuchter, M. (2021). Der Zusammenhang emotionaler Kosten bei Grundschullehramtsstudierenden mit ihrer informatischen Problemlösekompetenz, *35*(1), 93–111.

Weber, A. M., Bastian, M., Barkela, V., Mühling, A. & Leuchter, M. (2022). Fostering preservice teachers' expectancies and values towards computational thinking. *Frontiers in psychology*, *13*, 987761. https://doi.org/10.3389/fpsyg.2022.987761

Wildemann, A. & Merkert, A. (2020). *Sprachdiagnose, Sprachförderung und Sprachbildung in der Grundschule: Grundlagen, Methoden und Praxis* (1. Auflage). *Grundschule Deutsch*. Klett | Kallmeyer.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Zhang, L. & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, *141.* https://doi.org/10.1016/j.compedu.2019.103607

Karteikarten zu den Unterrichtssequenzen

Sequenz Nr.	Thema der Sequenz	Seiten
1	Befehle einführen und visualisieren anhand Calliope mini®	44 – 65
2	Algorithmen entwickeln anhand Calliope mini®	66 – 69
3	Fehler finden und beheben anhand Calliope mini®	70 – 74
4	Abstrahieren anhand Calliope mini®	75 – 77
5	Muster generalisieren anhand Calliope mini®	80 – 81
6	Probleme und Prozesse zerlegen anhand Calliope mini®	82 – 86

Befehle einführen und visualisieren anhand Calliope mini®
Übersicht

Kompetenzen

- Unterscheiden von Befehlen
- Beschreiben von Befehlen als Aktionen
- Zuordnen von Befehlen und Wirkungen
- Visualisieren erster Abfolgen von Befehlen

Wissen Lehrkraft

Befehle des Calliope verstehen und einsetzen können

- Icons
- Farben
- Wortlaut

Denk- und Arbeitsweisen einsetzen

- Vermuten
- Beobachten
- Ordnen
- Abfolgen erstellen
- Überprüfen

Materialien

Blatt mit drei Spalten

- Befehlsicons, -farben und -wortlaut
- Vermutung Bedeutung
- Beobachtung Wirkung

Befehle ausgeschnitten auf Karten

GEFÖRDERT VOM Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung
RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

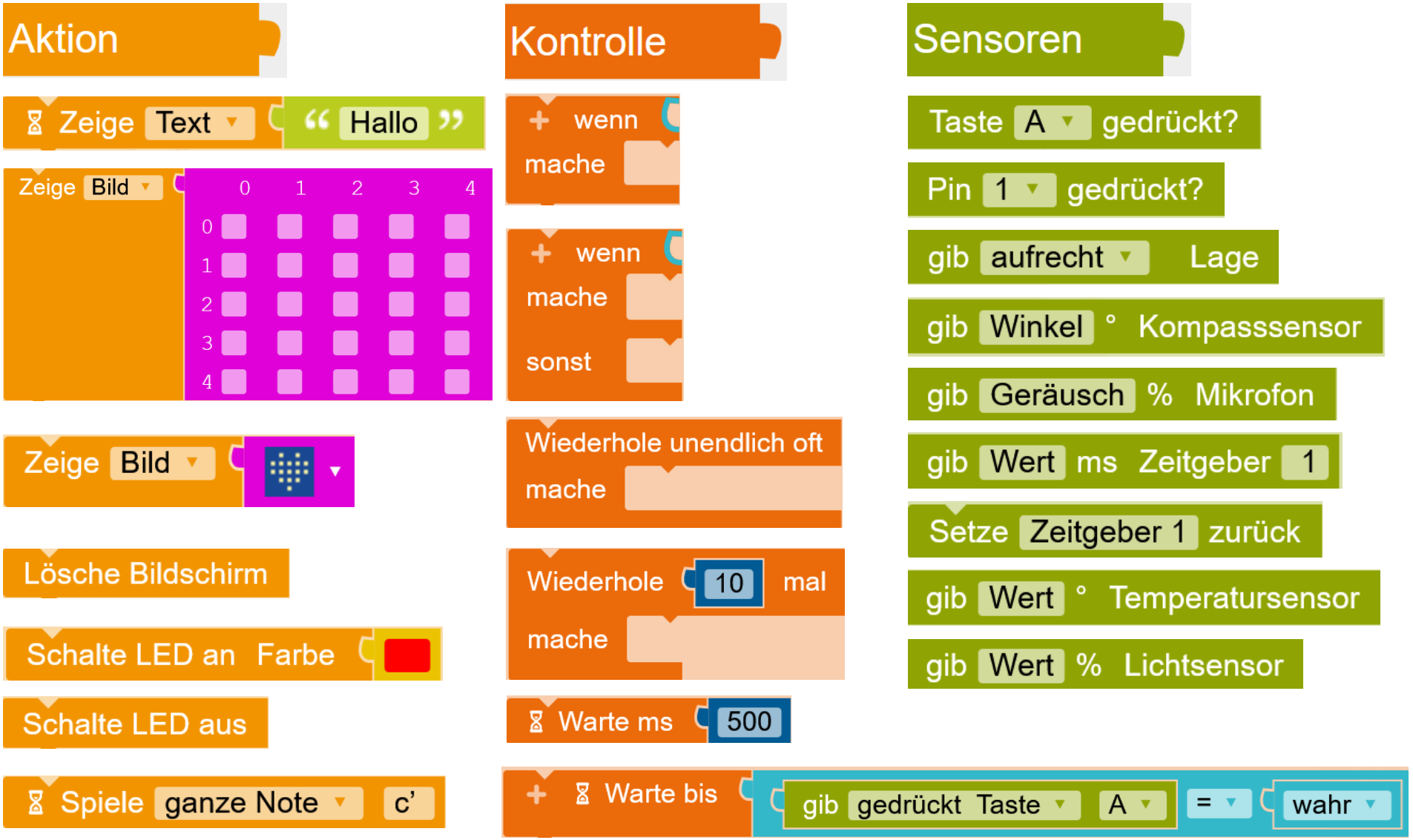
Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

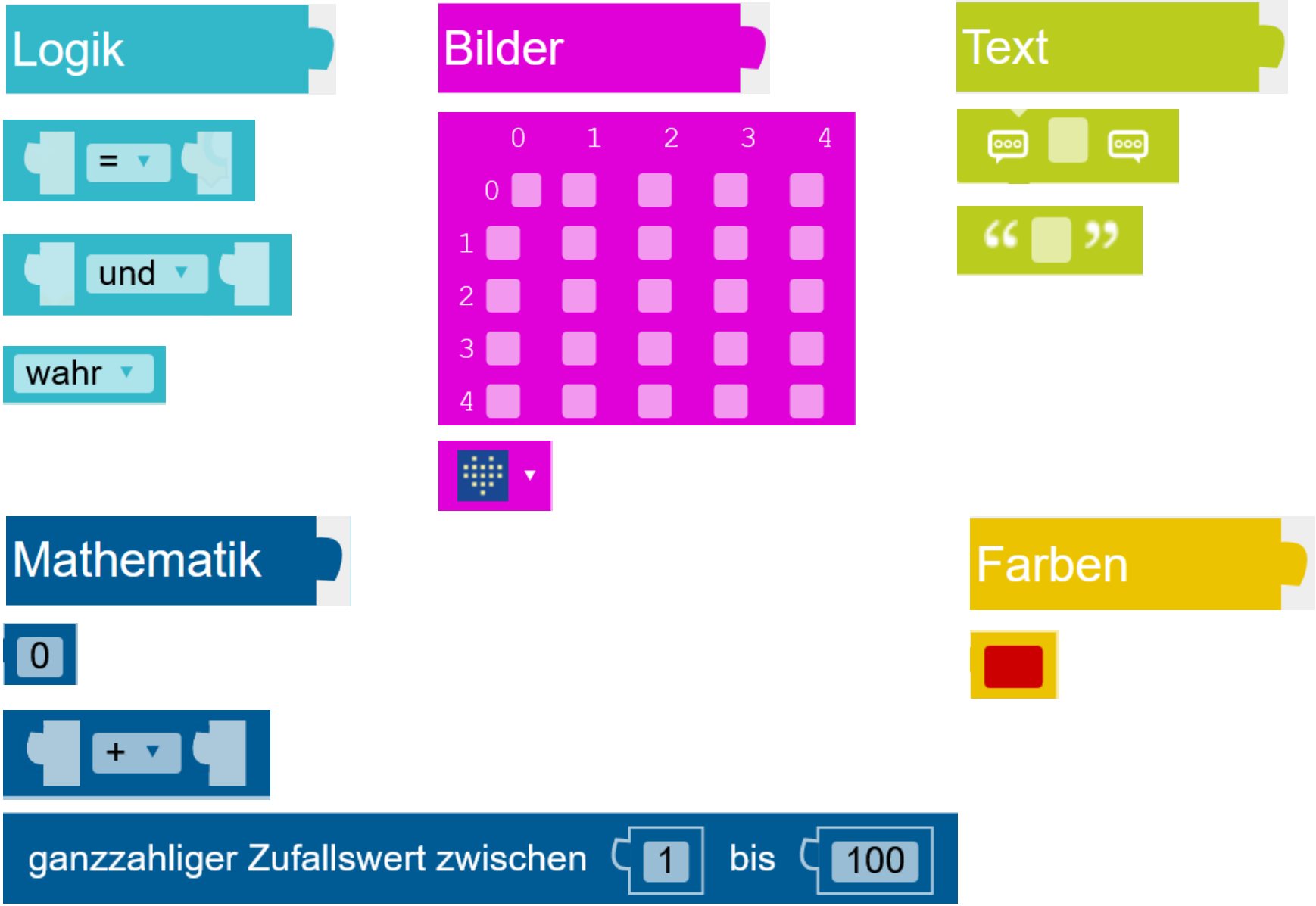
Befehle einführen und visualisieren anhand Calliope mini® Umsetzung

Kompetenz	Unterscheiden von Befehlen	Beschreiben von Befehlen	Zuordnen Befehle Wirkungen	Visualisieren von Abfolgen
Vorgehen Einzel-/ Partner-arbeit	Ausgeschnittene Befehle ordnen Memory spielen mit doppeltem Set	Aktionen vermuten auf Blatt Beschreiben der Vermutung	Überprüfen, wie die Befehle wirken Wirkungen auf Blatt festhalten Beschreiben der Wirkung	Abfolge legen Abfolge ausführen • Selber, ohne Werkzeug • Mit Werkzeug
Impulse	**Einführung** • Schau die Bildchen an • Du und dein Partner könnt damit Memory spielen **Aktivierung Vorwissen** • Hast du so etwas schon gesehen? **Begriffe** • Calliope • Befehl • Ordnen	**Einführung** • In der ersten Spalte siehst du die Bildchen • Vermute, welche Aktion sie auslösen • Schreib das in die zweite Spalte • Eine Vermutung ist nie falsch oder richtig **Begriffe** • Tabelle/Spalte/Zeile • Aktion • Vermutung	**Einführung** • Wo kannst du die Befehle einstellen? • Finde heraus, ob die Vermutung bestätigt / widerlegt wird. • Schreibe die Wirkung in die dritte Spalte • Beschreibe deinem Partner die Aktion **Begriffe** • Überprüfen • Bestätigen / Widerlegen	**Einführung** • Lege eine Befehlsabfolge • Dein Partner führt sie aus. • Befehlsabfolge am Tablet einstellen • Vergleicht Ausführung Partner und Roboter **Begriffe** • Befehlsabfolge • Ausführung • Vergleich

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau
Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.


Aktion
Zeige Text “ Hallo ”
Zeige Bild
0 1 2 3 4
0
1
2
3
4
Zeige Bild
Lösche Bildschirm
Schalte LED an Farbe
Schalte LED aus
Spiele ganze Note c’
Kontrolle
+ wenn
mache
+ wenn
mache
sonst
Wiederhole unendlich oft
mache
Wiederhole 10 mal
mache
Warte ms 500
+ Warte bis
gib gedrückt Taste A = wahr
Sensoren
Taste A gedrückt?
Pin 1 gedrückt?
gib aufrecht Lage
gib Winkel ° Kompasssensor
gib Geräusch % Mikrofon
gib Wert ms Zeitgeber 1
Setze Zeitgeber 1 zurück
gib Wert ° Temperatursensor
gib Wert % Lichtsensor

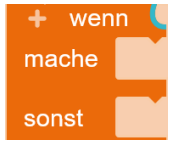
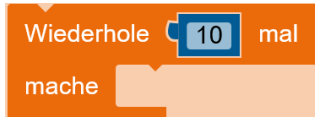
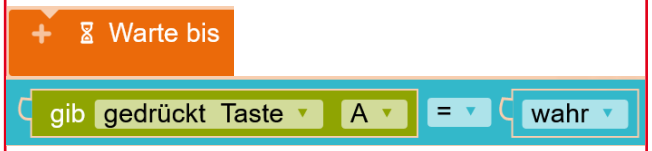


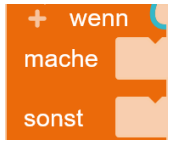
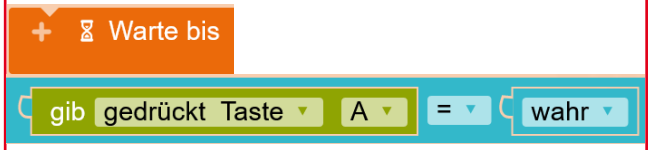
Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

Baustein	Wirkung
	
	
	
	
	
	
	

Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

Baustein	Wirkung
	Hier findest du die Aktionsbefehle.
	Zeige einen Text, der durchläuft. Es können Bausteine aus dem Text-Menü angehängt werden.
	Zeige Text in einzelnen Zeichen an. Es können Bausteine aus dem Text-Menü angehängt werden.
	Zeige ein Bild an. Es können Bausteine aus dem lila Bild-Menü angehängt werden.
	Lösche den Bildschirm.
	Schalte die LED an. Die Farbe kann ausgewählt werden.
	Spiele eine Note. Die Notenlänge und die Tonhöhe können ausgewählt werden.

Baustein	Wirkung
	
	
	
	
	
	
	

Baustein	Wirkung
	Hier findest du Kontroll-Bausteine.
	Setze diesen Baustein um andere Steine herum, damit Befehle immer wieder ausgeführt werden.
	Füge diesen Baustein in einen „Wiederhole“-Bausteine ein. An die „Wenn“-Bedingung kannst du einen Sensor-Baustein anhängen. An „Mache“ einen Aktions-Baustein.
	Dieser Baustein gibt an, was passieren soll, wenn die „Wenn“-Bedingung nicht erfüllt ist. Nutze ihn wie den „Wenn“-Baustein.
	Setze den Baustein um andere Steine, damit Befehle 10-mal ausgeführt werden. Die Zahl kannst du ändern.
	Calliope wartet eine bestimmte Anzahl Millisekunden, bis eine Aktion ausgeführt wird. Die Zahl kannst du ändern.
	Calliope wartet, bis eine bestimmte Aktion durchgeführt oder ein bestimmter Sensor betätigt wurde. An diesen Baustein kannst du Logik- und einige Sensoren-Bausteine anfügen.

Baustein	Wirkung
	
	
	
	
	

Baustein	Wirkung
	Hier findest du die Sensoren.
	Kann an „Wenn“-Bedingung oder in Logik-Bausteine angehängt werden. Du kannst Taste A oder B wählen. Drücke die Taste am Calliope, damit eine Aktion ausgeführt wird.
	Kann an „Wenn“-Bedingung oder in Logik-Bausteine angehängt werden. Du kannst zwischen den Pins wählen. Drücke die Pin und die entgegengesetzte am Calliope, damit eine Aktion ausgeführt wird.
	Kann an „Wenn“-Bedingung oder in Logik-Bausteine angehängt werden. Wähle die Lage und bringe Calliope in die Lage, damit eine Aktion ausgeführt wird.
	Kann in die Logik-Bausteine oder die Mathematik-Bausteine eingefügt werden. Gib einen Winkel an, um den Kompass-Sensor zu nutzen. Bringe Calliope in die Position, damit eine Aktion ausgeführt wird.

Baustein	Wirkung
	
	
	
	
	

Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

Baustein	Wirkung
	Kann in die Logik-Bausteine oder die Mathematik-Bausteine eingefügt werden. Gib eine Geräuschstärke, um den Geräusch-Sensor zu nutzen und eine Aktion auszulösen.
	Kann mit dem Aktion-Baustein „zeige Text“, den Logik-Bausteinen und den Mathematik-Bausteinen kombiniert werden. Zählt die Zeit, wenn Calliope eingeschaltet wird in Millisekunden.
	Setzt die Zeit auf 0 zurück.
	Kann in die Logik-Bausteine oder die Mathematik-Bausteine eingefügt werden. Gib eine Temperatur an, um den Temperatur-Sensor zu nutzen und eine Aktion auszulösen.
	Kann in die Logik-Bausteine oder die Mathematik-Bausteine eingefügt werden. Gib eine Lichtstärke an, um den Licht-Sensor zu nutzen und eine Aktion auszulösen.

Baustein	Wirkung
	
	
	
	

Baustein	Wirkung
	Hier findest du die Logik-Bausteine.
	Verwende diesen Baustein, wenn Du Grenzwerte bei Sensoren angeben willst/musst. Klicke auf das „=„ - Zeichen, um weitere Mathematik-Aktionen auszuwählen. Kombiniere ihn mit dem Zahlenbaustein.
	Dieser Baustein kann mit den Sensoren-Bausteinen kombiniert werden.
	Dieser Baustein wird mit dem Logik-Baustein kombiniert, um Aktionen auszuführen, wenn Bedingungen eintreten. Klicke auf „wahr“, um Aktionen auszuführen, wenn Bedingungen nicht eintreten.

Baustein	Wirkung
	
	
	
	

Baustein	Wirkung
	Hier findest du die Mathematik-Bausteine.
	Hier kannst du Zahlen eingeben. Dieser Baustein kann in Mathematik-Bausteine, Logik-Bausteine und in einige Kontroll-Bausteine eingefügt werden.
	Dieser Baustein kann mit dem Zahlen-Baustein verbunden werden. Klicke auf das Plus-Zeichen, um weitere Mathematik-Aktionen auszuwählen.
	Mit diesem Baustein zeigt Calliope einen Zufallswert an. Du kannst die Zahlen verändern.

Baustein	Wirkung
	
	
	

Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

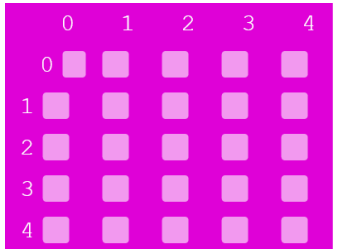
Baustein	Wirkung
	Hier findest du die Text-Bausteine.
	Hier kannst du einen Text eingeben. Füge den Baustein an den Aktions-Baustein: „Zeige Text“.
	Hier kannst du einen Kommentar schreiben, damit du später noch weißt, was du programmieret hast.

Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

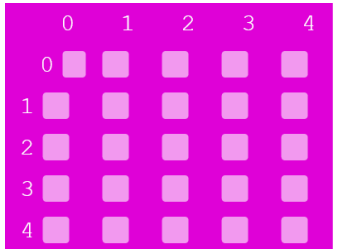
Baustein	Wirkung
	
	

Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

Baustein	Wirkung
	Hier findest du die Farben-Bausteine, für die LED.
	Du kannst weitere Farben auswählen, indem du auf den Baustein klickst. Der Baustein kann an den LED-Aktions-Baustein angehängt werden.

Baustein	Wirkung
	
	
	

Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

Baustein	Wirkung
	Hier findest du die Bilder-Bausteine.
	Mit diesem Baustein kannst du ein eigenes Bild kreieren, das Calliope anzeigt. Probiere verschiedene Tasten für verschiedene Farben: # ist rot, 1 ist gelb etc.
	Mit diesem Baustein kannst du vorgefertigte Bilder auswählen.

Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

Algorithmen entwickeln anhand Calliope mini®
Übersicht

Kompetenzen

- Befehlsabfolgen erstellen und programmieren
- Einfache Befehlsabfolgen gruppieren und als Bausteine verstehen
- Sequenzieren: Bausteine in Reihenfolge bringen
- Kontrollfluss: Reihenfolge der Bausteine evaluieren

Wissen Lehrkraft

Befehlsabfolgen als Bausteine verstehen

- Bausteine Sequenzieren
- Kontrollfluss evaluieren

Denk- und Arbeitsweisen einsetzen

- Beobachten
- Ordnen
- Planen
- Abfolgen erstellen
- Überprüfen

Materialien

- Befehle ausgeschnitten auf Karten
- Tesafilm

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

Algorithmen entwickeln anhand Calliope mini®
Übersicht

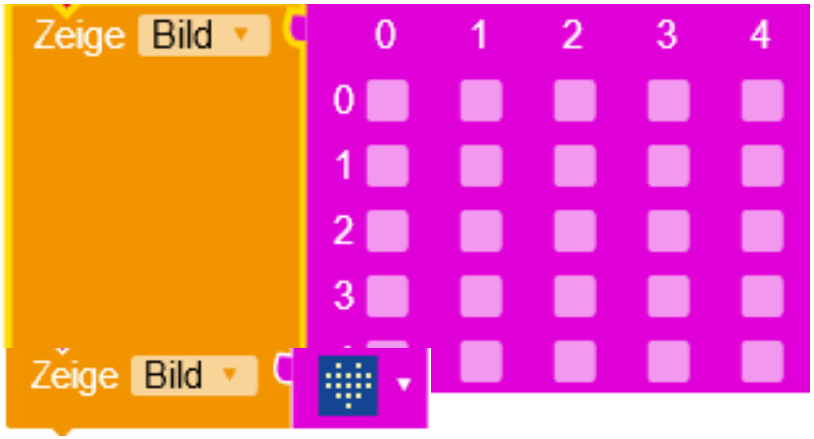
Kompetenz	Befehlsabfolgen erstellen	Befehlsabfolgen als Bausteine	Sequenzieren	Kontrollfluss
Vorgehen	Einfaches Ziel und Lösungsweg beschreiben Befehle suchen Befehle gruppieren	Befehle in Befehlsabfolgen überführen Gleiche Befehlsabfolgen erkennen	Überprüfen, wie die Bausteine wirken Abfolge Bausteine festlegen Abfolge überprüfen	Abfolge Bausteine programmieren Abfolge im Hinblick auf Problemlösung prüfen Abfolge korrigieren
Impulse	**Einführung** • Das ist das Ziel / das soll Calliope tun • Plane mit welchen Befehlen du das erreichen kannst • Übertrage dein Programm auf den Calliope und führe es aus **Begriffe** • Befehle • Ausführung • Beobachten • Programm	**Einführung** • Lege die Befehle so, wie dein Ziel erreicht werden konnte • Welchen Befehl musst du am Anfang immer setzen? • In welcher Reihenfolge müssen die Befehle stehen? • Klebe sie zusammen: das sind dann die Bausteine • Ordnet alle Bausteine, die gleich / anders sind **Begriffe** • Bausteine • Gleich / anders	**Einführung** • Hier ist ein Ziel. Es bedarf mehrerer Bausteine. • Welcher Befehl muss immer am sein? • In welcher Reihenfolge müssen die Befehle stehen? • Lege die Bausteine so, dass das Ziel erreicht wird • Überprüfe die Abfolge der Bausteine **Begriffe** • Bausteinabfolge • Planen • Überprüfen	**Einführung** • Formuliere ein eigenes Ziel und Teilziele als Abfolge • Du kannst nun Bausteine für Teilziele programmieren • Wird dein Ziel / Teilziel erreicht? • Korrigiere deine Bausteinabfolge **Begriffe** • Korrigieren

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau
Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.


Zeige Text
“ Hallo ”

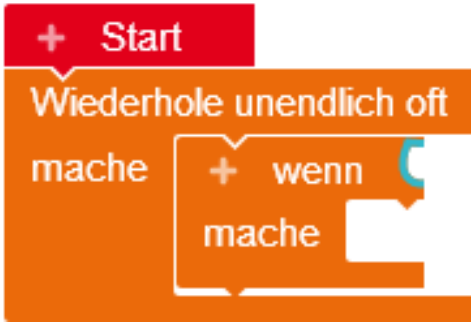

Zeige Zeichen
“ Hallo ”

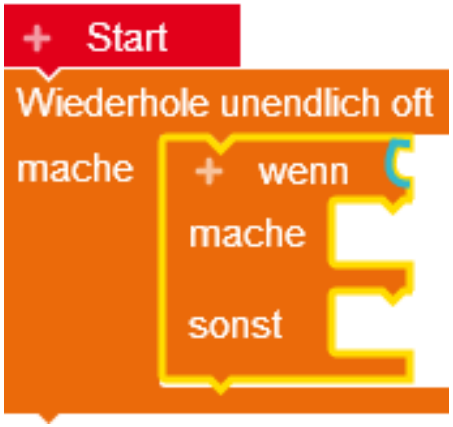

Zeige Bild
0 1 2 3 4
0
1
2
3
Zeige Bild


Schalte LED an Port intern Farbe

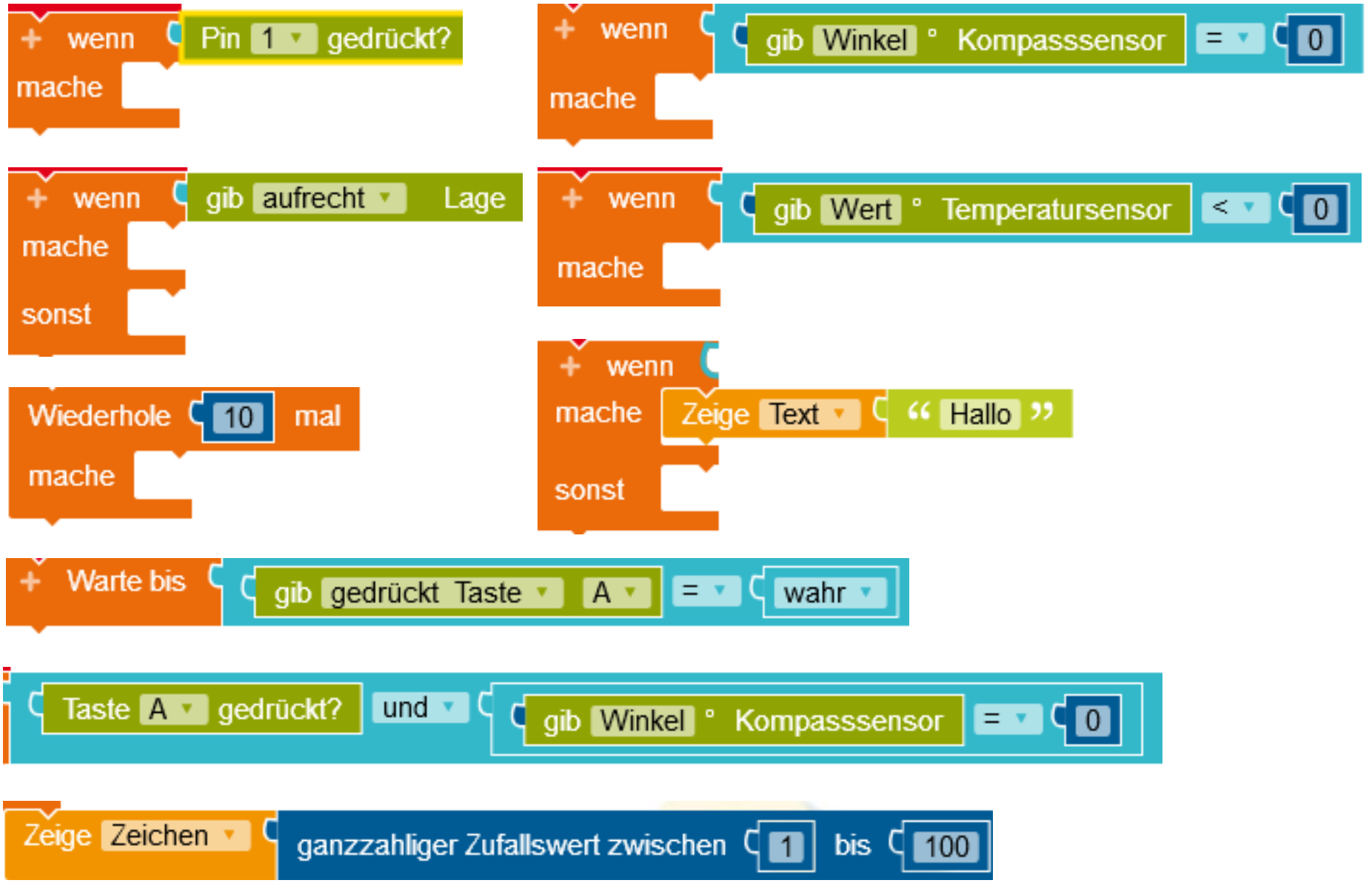

Start
Wiederhole unendlich oft
mache


Start
wenn
Taste A gedrückt?
mache


Start
Wiederhole unendlich oft
mache
wenn
mache


Start
Wiederhole unendlich oft
mache
wenn
mache
sonst


Warte ms
500



Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

Fehler finden und beheben anhand Calliope mini®
Übersicht

Kompetenzen

- Fehler erkennen
- Fehlerort erkennen identifizieren
- Art des Fehlers identifizieren
- Fehler beheben

Wissen Lehrkraft

Programmieren

- Befehle
- Bausteine
- Algorithmen

Materialien

- Ausgedruckte Befehle
- Ausgedruckte Bausteine

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

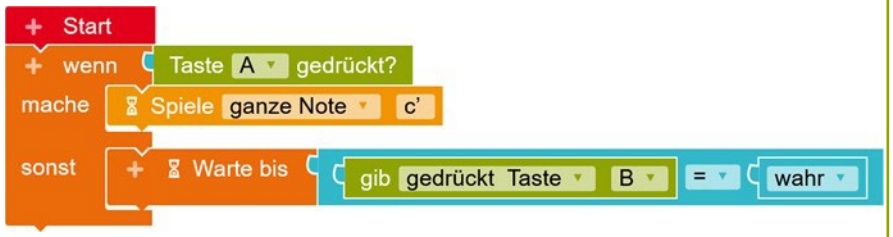
Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

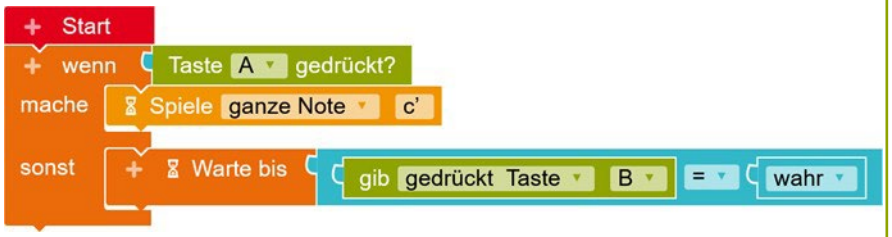
Fehler finden und beheben anhand Calliope mini®
Umsetzung

Kompetenz	Fehler erkennen	Fehlerort identifizieren	Art des Fehlers identifizieren	Fehler beheben
Vorgehen	Ein fertig programmiertes Problem einführen	Das Programm genau analysieren Das Programm mit Bausteinen / Befehlen auf Papier vergleichen	Die Bausteine / Befehle analysieren	Bausteine austauschen
Impulse	**Einführung** • Calliope soll wie eine Hupe funktionieren, wenn du auf A drückst. • Ich habe schon programmiert. Lass uns ausprobieren • Es funktioniert nicht. Wo liegt der Fehler? **Begriffe** • Ziel • Programmieren • Fehler	**Einführung** • Zerlege das Programm in Bausteine. • Überprüfe, was sie Bewirken. • Überlege, wo der Fehler liegen könnte. • Wurde etwas vergessen? • Ist ein falscher Baustein dabei? **Begriffe** • Bausteine • Befehle • Festhalten	**Einführung** • Was sollen die Bausteine bewirken? • Wie wird das Problem verursacht? • Liegt es an einem Baustein oder an mehreren? • Warum ist der sonst-Baustein zu viel? **Begriffe** • Wirkung	**Einführung** • Tausche die Bausteine auf dem Papier, an denen du denkst, dass es liegen könnte • Begründe • Überprüfe am Calliope **Begriffe** • Prüfen • Korrigieren • Austauschen

GEFÖRDERT VOM Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung
RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

Aufgabe	Ziel / Fehler
einfach 	
mittelschwer 	
schwer 	

Aufgabe	Ziel / Fehler
einfach 	„Wiederhole unendlich oft“ fehlt
mittelschwer 	„Wiederhole unendlich oft“ fehlt „sonst“ ist überflüssig, Programm erreicht aber trotzdem das Ziel
schwer 	„Wiederhole unendlich oft“ fehlt, „sonst“-Bedingung ist überflüssig (-> muss entfernt werden, oder Taste B in Taste A geändert werden)

Musterlösung

Abstrahieren anhand Calliope mini®
Übersicht

Kompetenzen

- Aufgabe / Problem verstehen
- Relevante Bausteine identifizieren
- Grundlegende Struktur eines Problems festhalten
- Komplexität minimieren

Wissen Lehrkraft

Aufgaben stellen

- Problemraum kennen
- Relevante Bausteine kennen
- Strukturen erkennen
- Komplexität minimieren

Denk- und Arbeitsweisen

- Kürzeste Problemlösung erkennen
- Korrigieren

Materialien

- Blatt
- Stift
- Ausgedruckte Befehle
- Ausgedruckte Bausteine

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau
Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

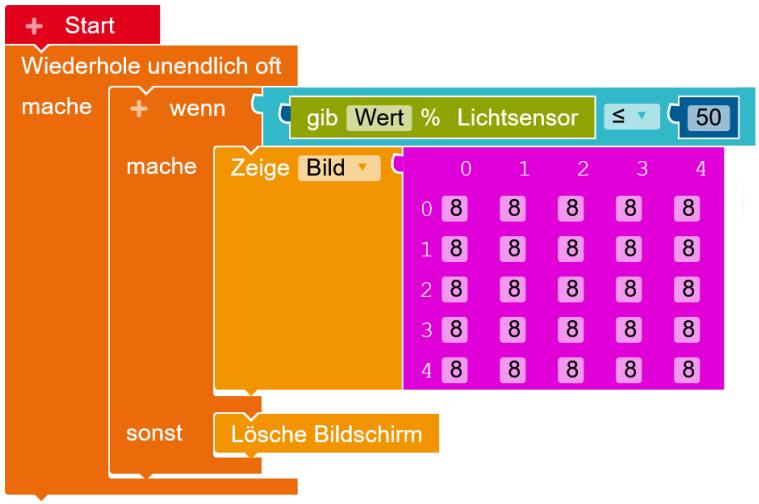
Abstrahieren anhand Calliope mini®
Umsetzung

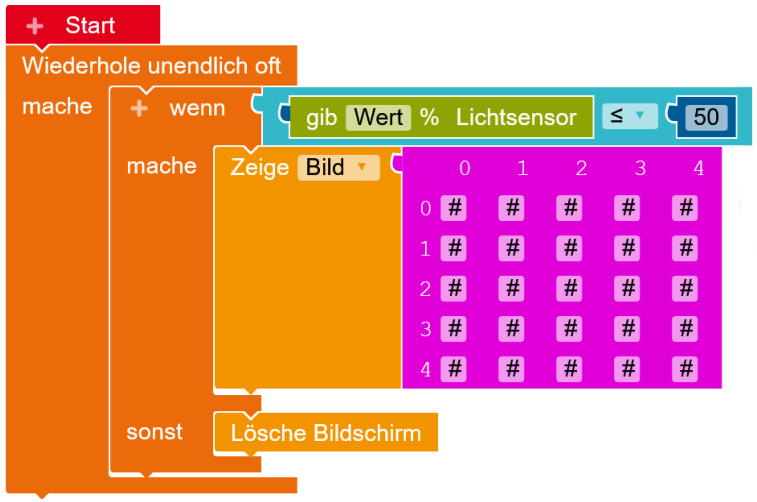
Kompetenz	Problem verstehen	Bausteine identifizieren	Struktur festhalten	Komplexität minimieren
Vorgehen	Ein Fahrradrücklicht vorstellen	Das Programm genau analysieren Das Programm mit Bausteinen / Befehlen auf Papier vergleichen	Mache eine Skizze / einen Plan Zeichne die Bausteine / Befehle Lege die Bausteine / Befehle	Abfolge im Hinblick auf Problemlösung prüfen Abkürzungen herausfinden Vereinfachen
Impulse	**Einführung** • Schau mal, ich habe mein Fahrradrücklicht mitgebracht. Lass uns das ausprobieren. • Können wir das mit Calliope selber programmieren? **Begriffe** • Ziel • Programmieren	**Einführung** • Aus welchen Kategorien (z.B. Kontrolle, Aktion, Logik) brauchen wir Bausteine? • Was bewirken die einzelnen Bausteine? • Hat Calliope einen Sensor? • Genau, manche Bausteine müssen an andere angehängt werden **Begriffe** • Bausteine • Befehle • Festhalten	**Einführung** • Was sollen die Bausteine bewirken? • Wofür brauche ich z.B. den „Wiederhole-unendlich-oft“-Baustein? • Wenn 0 dunkel ist und 10 hell, wann muss das Licht angehen, damit man dich auf dem Fahrrad sieht • Was kommt zuerst, was danach? **Begriffe** • Zuerst / danach • Übergang	**Einführung** • Meinst du, man könnte das auch anders lösen? • Kann man Bausteine weglassen/ersetzen? • Kann das Licht in einer anderen Farbe leuchten? **Begriffe** • Bausteinabfolge prüfen • Vereinfachen

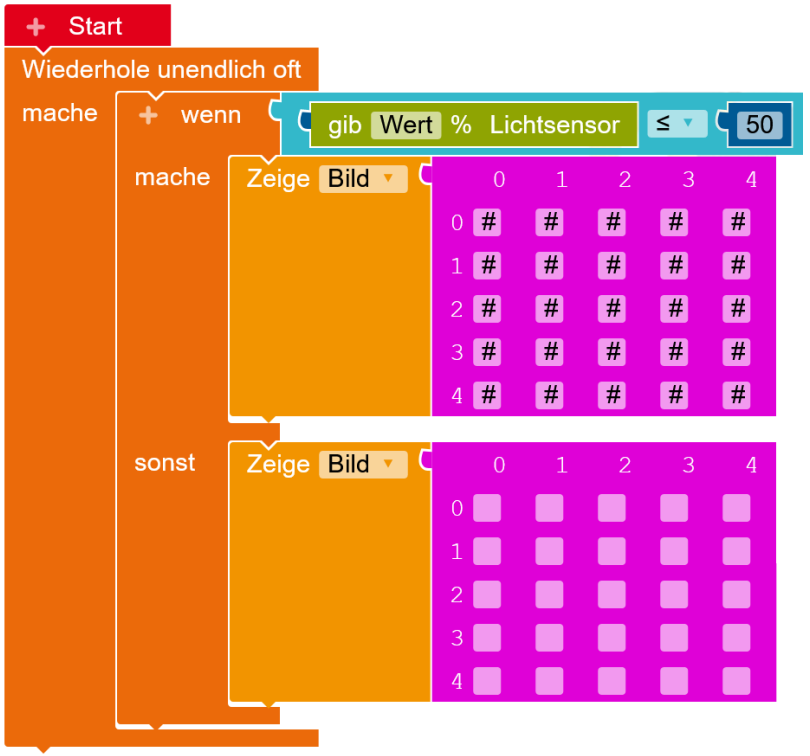
GEFÖRDERT VOM Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung
RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

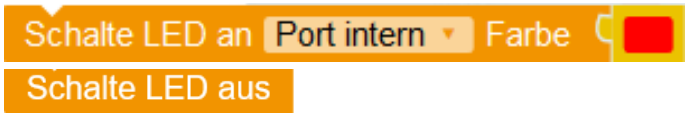
Lösungsmöglichkeiten







Es kann statt „zeige Bild“ auch die RGB-LED verwendet werden:



Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

Muster generalisieren anhand Calliope mini®
Übersicht

Kompetenzen

- Muster und Regelmäßigkeiten erkennen
- Regeln und Modelle entwickeln
- Regeln und Modelle überprüfen
- Verschiedene Probleme auf ein gemeinsames zugrundeliegendes Muster zurückführen

Wissen Lehrkraft

Programmieren

- Bausteine verstehen
- Programmmuster erkennen
- Regeln formulieren

Denk- und Arbeitsweisen

- Modellieren
- Generalisieren
- Überprüfen

Materialien

- Blatt
- Stift
- Ausgedruckte Befehle
- Ausgedruckte Bausteine

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

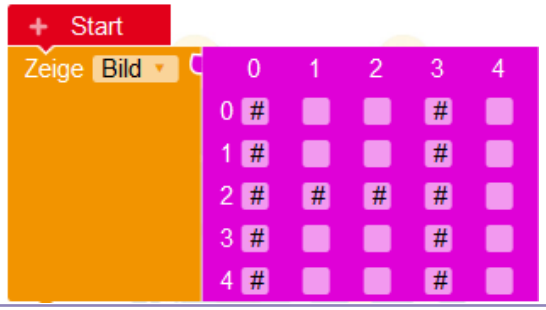
Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

Muster generalisieren anhand Calliope mini®
Umsetzung

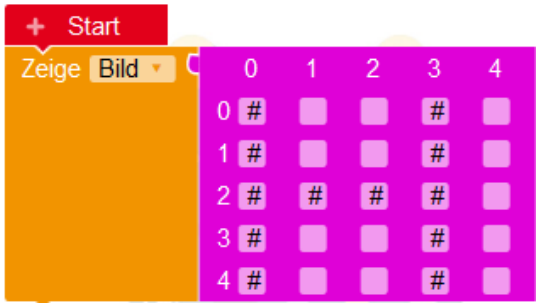
Kompetenz	Muster erkennen	Regeln entwickeln	Regeln überprüfen	Verschiedenes im Gemeinsamen
Vorgehen	Bausteine mit unterschiedlichen Befehlsabfolgen einführen, die das gleiche bewirken	Regeln für Bausteine und ihre Wirkung formulieren	Regeln überprüfen	Bausteine mit unterschiedlichen Befehlsabfolgen erstellen, die das gleiche bewirken
Impulse	**Einführung** • Hier habe ich unterschiedliche Bausteine. • Was ist gleich? • Was ist anders? • Was könnten sie bewirken? • Halte deine Vermutung fest **Begriffe** • Gleich / anders • Vermuten	**Einführung** • Halte fest, was gleich und was anders ist bei den Bausteinen • Warum bewirken sie das Gleiche, auch wenn sie anders sind? • Formuliere Regeln • Zeichne / schreibe Regeln auf **Begriffe** • Immer wenn – dann • Festhalten	**Einführung** • Stelle mal einen Baustein am einen Calliope ein • Stelle einen anderen Baustein am anderen Calliope ein • Bewirken sie das gleiche? • Korrigiere die Regeln **Begriffe** • Vergleichen • Überprüfen	**Einführung** • Wähle eine Regel • Erstelle selber Bausteine mit verschiedenen Befehlen, die das gleiche bewirken • Korrigiere **Begriffe** • Bausteinabfolge prüfen • Vergleichen

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau
Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

Muster	Vermutung
	
	
	

Erklärung:

Muster	Tatsächliche Wirkung
	Zeigt ein H auf dem LED-Display
	Zeigt ein H auf dem LED-Display
	Zeigt ein H auf dem LED-Display

Erklärung:

Die Anzeige des Buchstaben H kann als Zeichen, Text oder Bild programmiert werden. Der Befehl „Zeige“ bezieht sich immer auf den LED-Bildschirm, d.h. es werden LEDs aktiviert, um die Programmierung umzusetzen – in diesem Fall die LEDs, die ein H bilden. Es werden also bei allen drei Programmierungen die gleichen LEDs auf dem Bildschirm aktiviert.

Probleme und Prozesse zerlegen anhand Calliope mini®
Übersicht

Kompetenzen

- Probleme erkennen
- Prozesse formulieren
- Teilziele erkennen
- Teilprozesse programmieren

Wissen Lehrkraft

Problemraum erstellen

- Problem formulieren
- In Teilprobleme zerlegen
- Lösungsprozesse beschreiben

Denk- und Arbeitsweisen

- Beobachten
- Analysieren
- Planen
- Evaluieren

Materialien

Blatt mit drei Spalten

- Titel: Aufgabe
- Spalte 1: Ziel
- Spalte 2: Teilziel
- Spalte 3: Lösungswege für Teilziele

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau
Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

Probleme und Prozesse zerlegen anhand Calliope mini®
Umsetzung

Kompetenz	Problem / Ziel erkennen	Prozesse formulieren	Teilziele erkennen	Teilprozesse programmieren
Vorgehen	Aufgabe stellen Problemraum der Aufgabe erkennen	Mögliche Lösungswege erkennen Lösungswege frei formulieren	Problemraum in Teilprobleme zerlegen	Abfolge Bausteine programmieren Abfolge im Hinblick auf Problemlösung prüfen Abfolge korrigieren
Impulse	**Einführung Spalte 1** • Calliope soll ein umschaltbares Fahrradlicht sein, das blinken und durchgehend leuchten kann. • Was kennst du schon • Was ist schon da • Was ist das Ziel • Was willst du mit der Aufgabe erreichen • Zeichne / Schreib auf **Begriffe** • Erklären • Vermuten • Planen	**Einführung** • Kannst du einige Bausteine erkennen, die gebraucht werden? • Welche Lösungswege siehst du? • Zeichne/Schreibe sie auf **Begriffe** • Planen • Lösungswege	**Einführung Spalte 2** • Kannst du die Aufgabe in Teile zerkleinern / zerlegen? • Was ist zuerst / später, einfacher / schwieriger? • Wie kannst du das Blinken programmieren? • Was brauchst du zum Umschalten? **Begriffe** • Ziel / Teilziel • Als erstes / später • Einfach / schwierig • Zerlegen / Zerkleinern	**Einführung Spalte 3** • Du kannst nun Bausteine für Teilziele programmieren • Lege die Bausteine gemäß der Abfolge der Teilziele • Wird dein Ziel / Teilziel erreicht? • Korrigiere deine Bausteinabfolge **Begriffe** • Prüfen • Korrigieren

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
MoSAiK FoSach
Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau
Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

Aufgabe: Calliope soll leuchten wie ein Fahrradlicht, wenn es dunkel wird.

Ziel der Aufgabe	Teilziele	Lösungswege für Teilziele

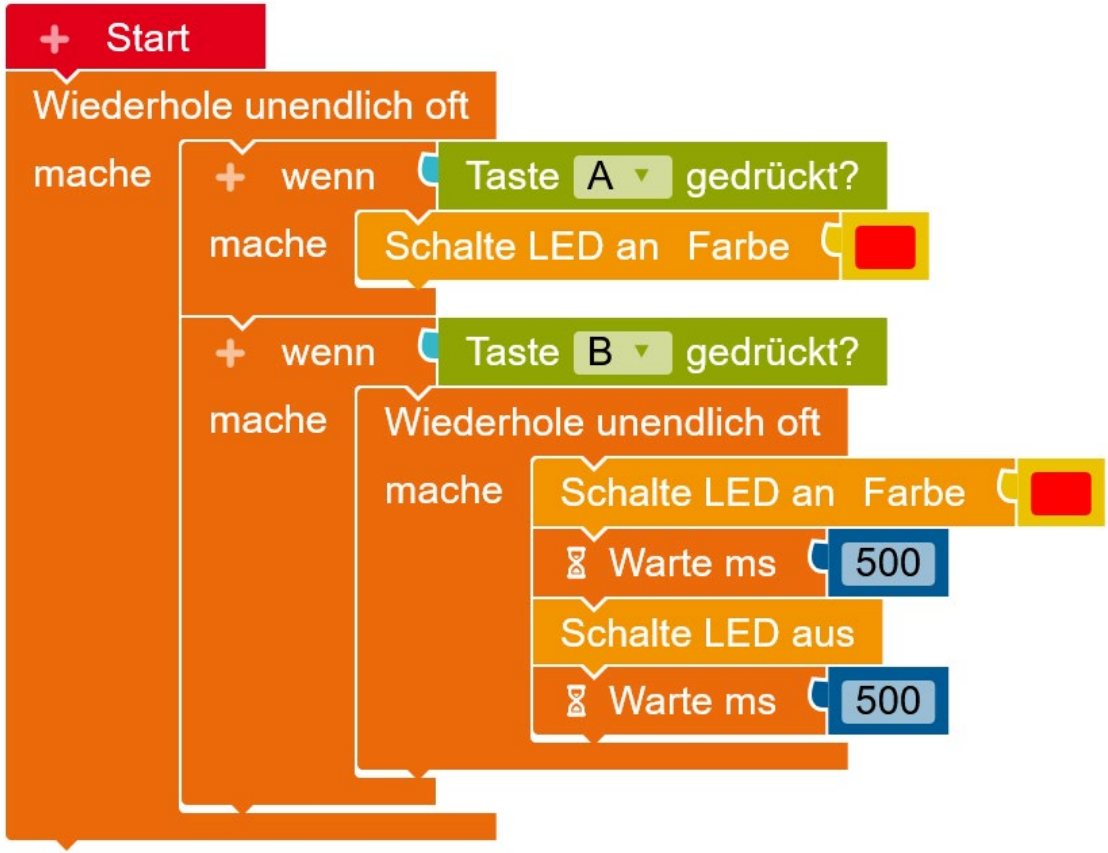
GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
MoSAiK FoSAch
Deutsche Telekom Stiftung

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

Weber, A.; Barkela, V.; Stiel-Dämmer, S.;
Ebel, M.; Reuter, T.; Leuchter, M.

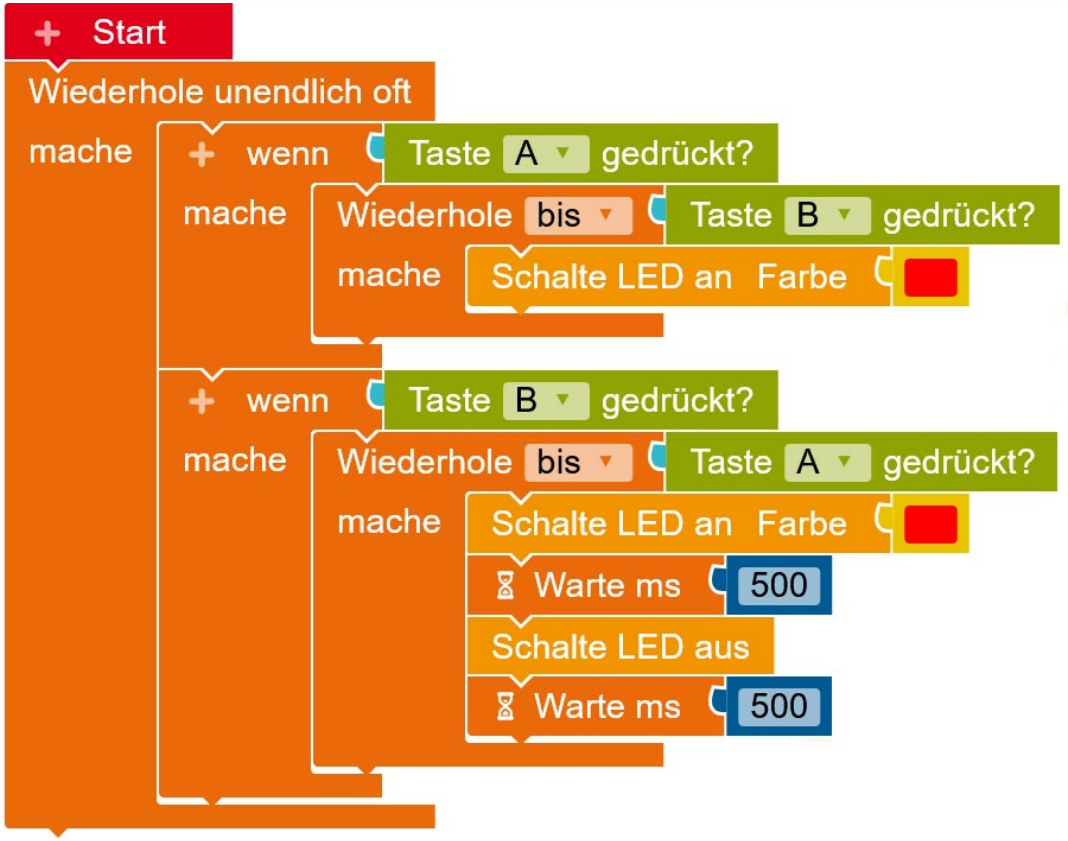
Lösungsmöglichkeiten

Leichtere Variante



Lösungsmöglichkeiten

Schwierige Variante (Baustein „Warte bis“ gehört zum „Experten-Level“)



Weber, A.; Barkela, V.; Stiel-Dämmer, S.; Ebel, M.; Reuter, T.; Leuchter, M.

Autor*innen

Dr. Maren Ebel ist wissenschaftliche Mitarbeiterin am Forschungsschwerpunkt Sachunterricht der RPTU. Ihre Forschungsinteressen sind Förderung von naturwissenschaftlichen, geographischen und bildungssprachlichen Kompetenzen im Vor- und Grundschulalter.

Veronika Barkela ist wissenschaftliche Mitarbeiterin am Forschungsschwerpunkt Sachunterricht der RPTU. Ihre Forschungsinteressen sind kognitive und motivationale Prozesse beim Lernen und Lehren von Computational Thinking und in computerbasierten Lernumgebungen sowie Selbstregulationsprozesse beim Umgang mit automatisiertem Feedback.

Dr. Timo Reuter ist wissenschaftlicher Mitarbeiter am Forschungsschwerpunkt Sachunterricht der RPTU. Seine Forschungsinteressen sind Lehr-Lernprozesse im Vor- und Grundschulalter bei naturwissenschaftlichen und technischen Themen, Problemlösen sowie hochschuldidaktische Fragestellungen.

Sabrina Stiel-Dämmer ist wissenschaftliche Mitarbeiterin am Forschungsschwerpunkt Sachunterricht der RPTU. Ihre Forschungsinteressen sind Lehr-Lernprozesse im Vor- und Grundschulalter bei naturwissenschaftlichen und technischen Themen sowie professionelle Kompetenzen von Studierenden, Pädagogischen Fachkräften und Lehrkräften.

Prof. Dr. Miriam Leuchter Leiterin des Forschungsschwerpunkts Sachunterricht der RPTU. Ihre Forschungsinteressen sind frühes Lernen in Naturwissenschaften und Technik, digitale Bildung sowie Professionelle Kompetenzen von Studierenden, Pädagogischen Fachkräften und Lehrkräften.

Dr. Anke Weber ist Postdoctoral Researcher im Department of Behavioural and Cognitive Sciences der University of Luxembourg. Ihre Forschungsinteressen sind die Entwicklung des räumlichen Denkens, analogen und kausalen Schlussfolgerns bei jungen Kindern, Eltern-Kind-Spiel und emotionale sowie motivationale Prozesse beim Lernen und Lehren von Computational Thinking.