Programmierunterricht von Kindergarten bis zur Matura in einem Spiralcurriculum

Jacqueline Staub · Michelle Barnett
Nicole Trachsler

Einführung

In der Schule lernen Kinder nebst lesen und schreiben nun auch zu programmieren. Dabei werden Fähigkeiten wie Kreativität und konstruktives Problemlösen gefördert. Diese sind wichtige Voraussetzungen, um sich in den noch ungewissen Berufen der Zukunft behaupten zu können. Wer schon in der Schule übt, allgemeine Probleme zu lösen und Lösungswege exakt zu beschreiben (so exakt, dass selbst intellektfreie Maschinen sie für uns ausführen können), ist gut vorbereitet, die noch unbekannten Herausforderungen der Zukunft bewältigen zu können. Guter Programmierunterricht erlaubt es, die Lernenden gezielt darauf vorzubereiten.

Was bedeutet programmieren?

Programmieren bedeutet, im engeren Sinne, mit einer Maschine zu kommunizieren. Dazu müssen wir eine Sprache verwenden, die die Maschine versteht: eine **Programmiersprache**. Ähnlich wie natürliche Sprachen (Deutsch, Französisch oder Englisch) besteht auch eine Programmiersprache aus Wörtern. Diese nennt man in der Informatik **Befehle**. Durch die grammatikalisch korrekte Aneinanderreihung von Befehlen bilden sich größere Texte, die wir als **Programme** bezeichnen. Programmieren verlangt einen hohen Grad an Präzision: Um den Computer anzuleiten, eine Tätigkeit auszuführen, müssen wir Schritt für Schritt detailliert beschreiben, was zu tun ist. Der Computer besitzt keinerlei Improvisationsfähigkeit, weshalb die Beschreibung vollständig und eindeutig interpretierbar sein muss. So lernen Kinder, sich formal und logisch korrekt auszudrücken.

Wieso programmieren wir?

Programmieren ist eine inhärent konstruktive Tätigkeit: Lösungen werden konstruiert, modular kombiniert und erweitert. Im Programmierunterricht wird die konstruktive Denkweise der technischen Disziplinen gefördert. Die folgenden drei Punkte erachten wir als zentrale Ziele des Programmierunterrichts [4]:

1. Mittels Probieren und Experimentieren Lösungswege für gegebene Aufgabestellungen entdecken
2. Gefundene Lösungswege mittels Programmieren umsetzen und funktionsfähige Programme daraus entwickeln
3. Die korrekte Funktionalität durch Ausführen des Programms am Computer testen und nach Bedarf das Programm korrigieren oder es für zusätzliche Aktivitäten erweitern

Die Kinder lernen im Unterricht selbstständig Neues zu entdecken, ihre eigenen sowie gegebene Vorgehensweisen kritisch zu hinterfragen und gegebenenfalls zu verbessern. Sie werden zu kreativen Produzenten, die nach eigenem Empfinden neue und innovative Lösungen entwickeln.

Unser Spiralcurriculum

Zwischen Kindergarten und Maturitätsreife wächst die Bandbreite der erlangten Kompetenzen stetig,

https://doi.org/10.1007/s00287-019-01161-6

Jacqueline Staub · Michelle Barnett · Nicole Trachsler
ETH Zürich
E-Mail: {staubj, barnettm, tnicole}@ethz.ch

Jacqueline Staub
PH Graubünden

Zusammenfassung

In diesem Artikel stellen wir drei Programmierumgebungen vor, welche an der ETH Zürich entwickelt wurden und in einem einheitlichen Spiralcurriculum für den Programmierunterricht vom Kindergarten bis zur Maturität verwendet werden können. Der Fokus liegt auf der selbstständigen Entwicklung funktionsfähiger Programme. Ausgehend von einer Aufgabenstellung beginnt der Prozess der Erkenntnisgewinnung mittels Ausprobierens, Entwickeln einer Lösungsstrategie und deren Umsetzung in einem vollständigen Programm. Dieses wird schließlich getestet, indem es am Computer ausgeführt und nach Bedarf verbessert oder erweitert wird. Unsere Programmierumgebungen unterstützen hohe Selbstständigkeit der Kinder und Jugendlichen in allen Phasen der Herstellung eines Produktes im Sinne eines funktionierenden Programmes.

vom Lesen und Schreiben bis hin zu abstraktem Denken. Unsere Materialien (das Curriculum ebenso wie die Lernumgebungen) berücksichtigen diese kognitive Entwicklung und knüpfen gezielt an den Wissensstand der Lernenden an.

In den folgenden Kapiteln stellen wir diese kognitive Entwicklung vor. Wir illustrieren zentrale Herausforderungen, die in den jeweiligen Stufen auftreten, und erklären, wie unsere Umgebungen die Lernenden darin unterstützen, diese Herausforderungen zu meistern.

Programmiere die Biene

Schon ab dem Kindergarten kann das Lösen einfacher Problemstellungen mittels Programmieren gefördert werden. Dazu stehen diverse kleine, robuste und einfach zu bedienende Roboter (wie z. B. der BeeBot) zur Verfügung, die sich auf Befehl hin steuern lassen. Auf Bodenmatten (wie in Abb. 1) lösen Kinder Navigationsaufgaben. Spielerisch lernen sie dabei ihre erste Programmiersprache. Um den Roboter zu steuern, stehen vier einfache Befehle zur Verfügung: Diese navigieren den Roboter ein Feld vor (respektive zurück) oder drehen ihn am Ort um 90 Grad nach rechts (respektive links).

Blockbasiertes Programmieren am Computer

Anknüpfend an die Erfahrungen mit dem Bienenroboter steht die von uns entwickelte Programmierumgebung XLogoBlocks (https://xlogo.inf.ethz.ch/1) zur Verfügung. Darin steuern die Lernenden am Bildschirm eine Schildkröte, die eine Spur hinterlässt und so farbige Muster auf der Zeichenfläche erzeugt (s. Abb. 2).

Für Anfänger allgemein, aber insbesondere für Anfänger dieser Altersstufe, gilt es, die verwendete Programmiersprache und deren zugrunde liegenden Befehle sorgfältig auszuwählen unter Rücksichtnahme auf die verfolgten didaktischen Ziele. In dieser Stufe wollen wir das planende Denken sowie die räumliche Vorstellungskraft der Lernenden stärken. Aus diesem Grund offeriert die Umgebung bewusst nur wenige Befehle, die vom BeeBot mehrheitlich bereits bekannt sind.

Abb. 1 BeeBot und Matte [8]

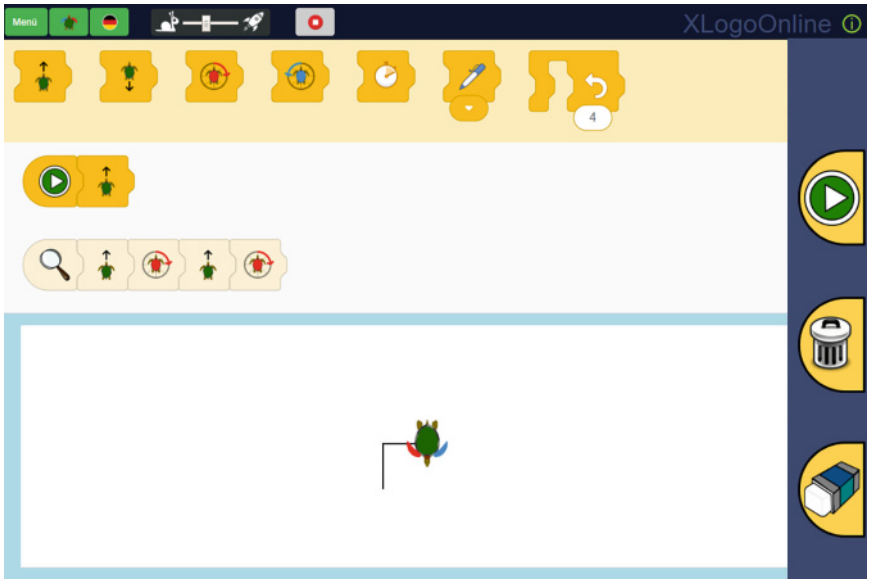


Abb. 2 XLogoBlocks

Sämtliche Befehle werden durch Blöcke dargestellt. Das **blockbasierte Programmieren** eignet sich besonders für junge Kinder, deren sprachliche und feinmotorische Fähigkeiten noch nicht ausgereift genug sind, um auf einer Tastatur zu arbeiten. Dank symbolischer Darstellung der Befehle werden weder Lese- noch Schreibfähigkeiten vorausgesetzt. Tippfehler sind ausgeschlossen und die Kinder konzentrieren sich auf das Erstellen logisch korrekter Programme.

Dazu werden die Befehle wie Puzzleteile mit der Maus zum Startblock gezogen und dort auf Knopfdruck ausgeführt. Bereits ausgeführte Befehle oder Befehlssequenzen werden gesammelt und in einer Übersicht veranschaulicht (s. Abb. 3). Diese zeigt den bisherigen Lösungsweg als Programm und entspricht dem von der Schildkröte gelaufenen Muster. Dank Übersicht kann der Lösungsweg nachvollzogen, diskutiert und nach Bedarf für Korrekturen verwendet werden.

Korrekte Programme zu schreiben, ist nicht einfach. Denn die Kinder vollziehen beim Programmieren einen ungewohnten Perspektivenwechsel: Sie versetzen sich in die Lage der Schildkröte. Dass deren Perspektive nicht immer mit der eigenen übereinstimmt, stellt viele Lernende vor eine kognitive Herausforderung. Wir unterstützen sie durch eine visuelle Hilfestellung: Die Arme der Schildkröte sind unterschiedlich eingefärbt – in derselben Farbe wie die entsprechenden Befehle *rechts* und *links* (s. Abb. 4).

Kontinuierlich lernen die Kinder, sich in einer neuen Sprache auszudrücken: Sie lesen Programme (i. e. prognostizieren deren Effekt) und beschreiben eigene Lösungswege als Sequenzen von Befehlen. Anfangs werden Lösungen Schritt für Schritt konstruiert. Später werden auch längere Befehlssequenzen geschrieben und ausgeführt.

Die Komplexität der Aufgaben kann beliebig angepasst werden. Abbildung 5 zeigt zwei Programme, die je ein Quadrat in unterschiedlicher Größe auf den Bildschirm zeichnen.

Jeder Schritt der Schildkröte hat eine konstante Länge. Um längere Linien zu zeichnen, müssen meh-

Abb. 3 Bereits ausgeführte Befehle

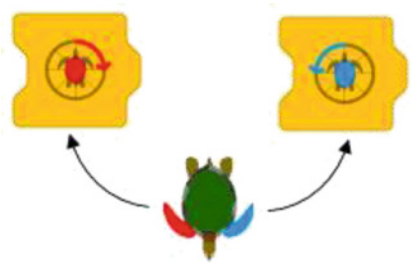


Abb. 4 Perspektivenwechsel

Abb. 5 Zwei Programme, welche Quadrate verschiedener Größen zeichnen

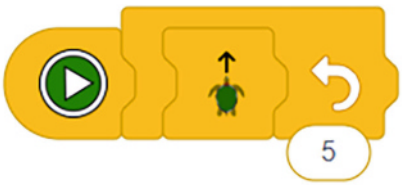


Abb. 6 Der neue Befehl Repeat

rere Schrittbefehle aneinandergereiht werden (wie im unteren Programm der Abb. 5). Mit dem Befehl *Repeat* ermöglichen wir den Kindern, lange Linien mit nur wenigen Befehlen zu zeichnen. Zur Veranschaulichung: Das Programm in Abb. 6 lässt die Schildkröte fünf Schritte vorwärtsgehen, benötigt dazu jedoch nur zwei Befehle.

Kinder im Alter zwischen fünf und acht Jahren machen gerade erste Erfahrungen mit Zahlen. Durch die Einführung des Befehls *Repeat* erhalten die Kinder die Möglichkeit, die Schildkröte mehr als nur einen Schritt aufs Mal vorwärtsgehen zu lassen. Die Kinder zählen die Schritte der Schildkröte und vertiefen so indirekt ihr Verständnis für Zahlen.

Variable Distanzen und Winkel

Mit zunehmendem Fortschritt erlangen Kinder etwa ab der dritten Klasse ein intuitives Verständnis für den Zahlenraum bis 1000 und die grundlegenden Rechenoperationen darin.

An dieser Stelle verallgemeinern wir die verfügbaren Befehle (Abb. 7) in XLogoBlocks (https://xlogo.inf.ethz.ch/2). Neu können die Befehle durch sogenannte **Parameter** modifiziert werden. Die Kinder geben bei Bewegungsbefehlen an, wie lange eine zu zeichnende Linie sein soll. Bei Rotationsbefehlen wird entsprechend angegeben, um welchen Winkel sich die Schildkröte drehen soll.

Diese Änderung erlaubt es den Kindern eine Vielzahl neuer Muster zu zeichnen. Sämtliche Quadrate (unabhängig von deren Größe), können mittels

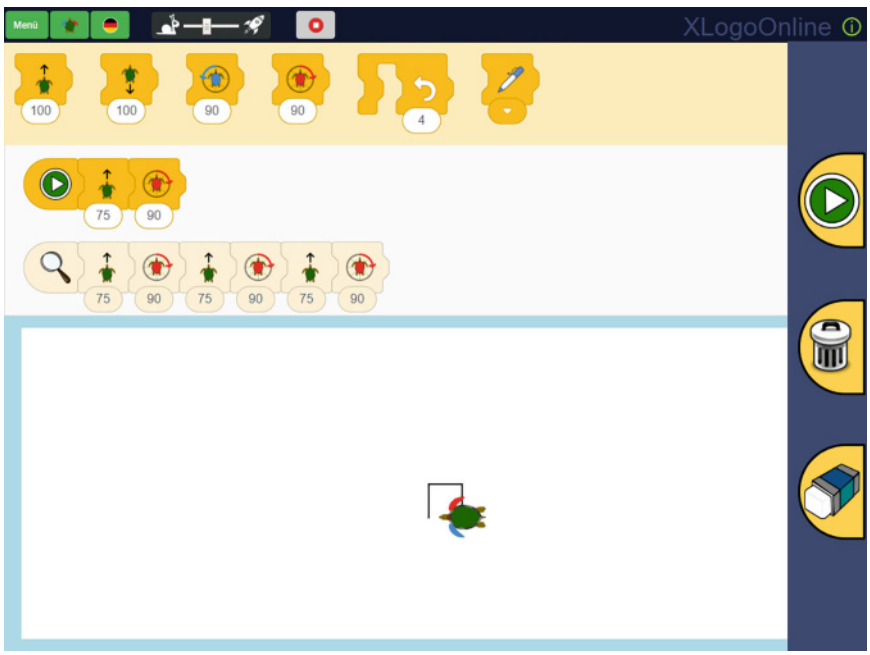


Abb. 7 Neue Befehle in XLogoBlocks

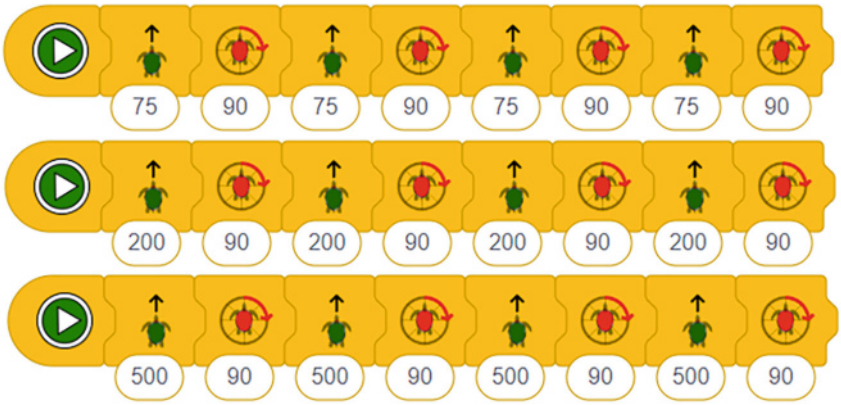


Abb. 8 Quadrate verschiedener Größen

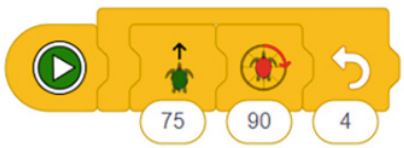


Abb. 9 Quadrat mit Repeat

Parameter durch nur acht Befehle dargestellt werden (siehe Abb. 8).

Mit steigender Erfahrung erkennen Kinder bald wiederkehrende Programmsequenzen in ihren Lösungswegen. Zum Zug kommt erneut der Befehl *Repeat*, mit welchem die Kinder lernen, lange und unübersichtliche Programme kürzer zu beschreiben. Anders als in der vorgehenden Stufe, werden hier nicht nur einzelne Befehle, sondern ganze Befehlssequenzen wiederholt. Dazu müssen die Lernenden wiederkehrende Muster in ihren Programmen identifizieren und dieses ausformulieren. Das Zeichnen eines Quadrats besteht z. B. aus vier identischen Seiten: Eine Linie und eine Drehung um 90°. Mit dem Befehle *Repeat* lässt sich das kurz beschreiben (Abb. 9).

Das Konzept der Repetition stellt ein mächtiges Werkzeug dar, um Arbeit durch den Computer zu automatisieren. Es schafft die Ausgangslage um in der nächsten Stufe den Mechanismus der Modularität zu verstehen.

Textbasiertes Programmieren und Modularität

Geschriebener Text hat sich über Jahrtausende als einer der beliebtesten Informationsträger etabliert. Auch im professionellen Umfeld werden Programme stets in Textform verfasst. Dies verlangt jedoch Präzision, denn Tippfehler werden von unserem intellektfreien Kommunikationspartner nicht verstanden. Circa ab der fünften Klasse haben die Kinder ausreichende Feinmotorik- und Sprachkompetenzen entwickelt, um diese Herausforderung zu bewältigen und dabei zu lernen, sich nicht nur logisch, sondern auch formal präzise auszudrücken.

Wir verwenden die Programmiersprache Logo, welche 1967 vom Mathematiker und Psychologen Seymour Papert und seinem Team entwickelt wurde. Logo umfasst sämtliche der bisher vorgestellten Befehle. In der folgenden Liste zeigen wir auf, wie die bereits bekannten Befehle in Logo geschrieben werden (Abb. 10).

Die Programmierumgebung XLogoOnline (https://xlogo.inf.ethz.ch/3, siehe Abb. 11) ermöglicht den Einstieg ins textbasierte Programmieren.

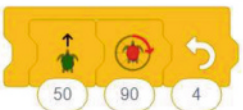
Blockbasierter Befehl	Textbasierter Befehl (Kurzform)
	forward 50 (fd 50) back 50 (bk 50)
	right 90 (rt 90) left 90 (lt 90)
	repeat 4[fd 50 rt 90]

Abb. 10 Befehle in Logo

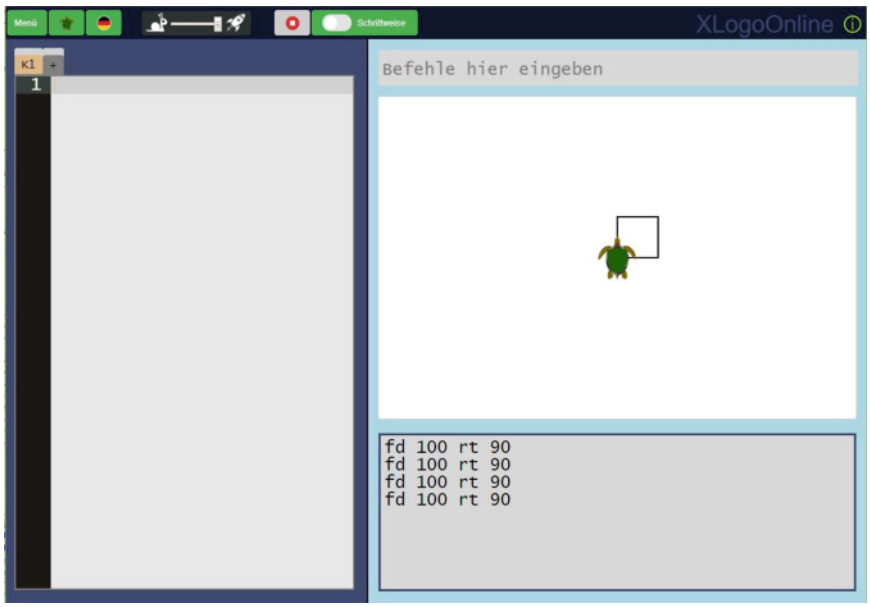


Abb. 11 XLogoOnline

Die Umgebung ist auf Programmieranfänger ausgerichtet und unterstützt ihren Lernprozess mittels gezielter Hilfestellung bei Fehlern, was sie von anderen Programmierumgebungen abhebt [7].

Programmieren in geschriebener Sprache erlaubt es den Kindern, beliebig große und komplexe Muster zu erzeugen. Gleichzeitig wird der Prozess jedoch auch anfälliger für Fehler: Jeder Tippfehler, sei er noch so unauffällig und klein, führt dazu, dass der Computer die Ausführung des Programms verweigert. XLogoOnline unterstützt die Programmierenden gezielt darin, mit dieser Herausforderung umzugehen und die Fehler selbstständig zu beheben. Die Umgebung bietet zwei Arten von Hilfestellungen. Diese betreffen zwei Typen von Fehlern: **syntaktische Fehler** und **logische Fehler**.

1. Enthält das Programm syntaktische Fehler, wird es vom Computer nicht verstanden. Vergisst

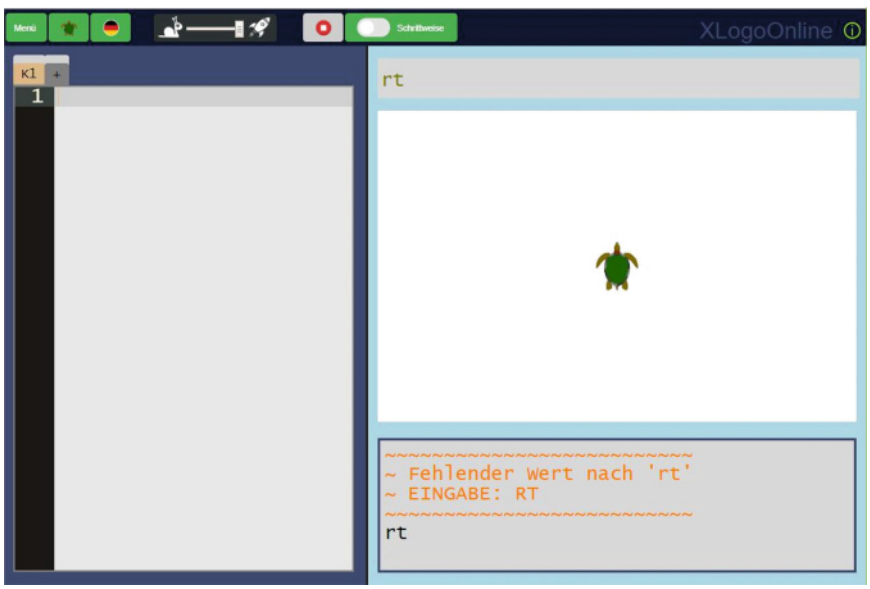


Abb. 12 Fehlermeldungen

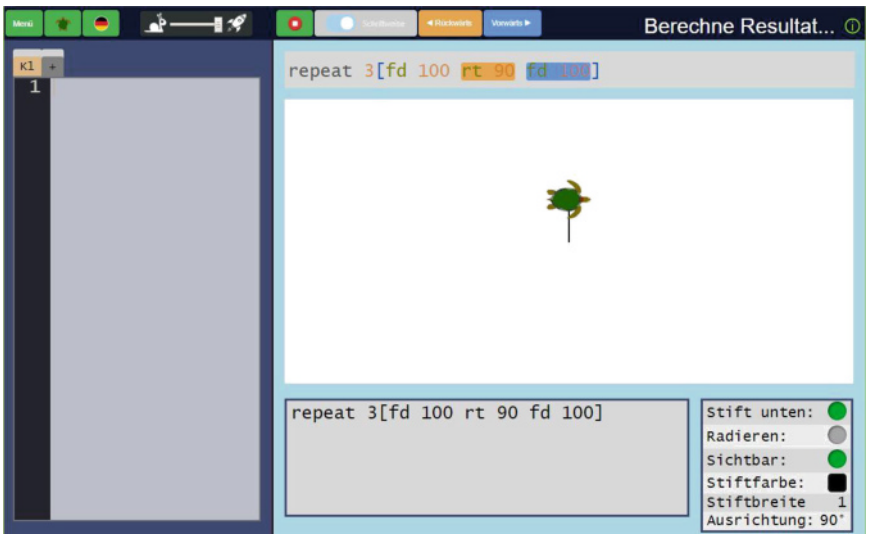


Abb. 13 Schritt für Schritt wird das Programm nachvollzogen

Abb. 14 In Orange der zuletzt ausgeführte und in Blau der nächste auszuführende Befehl

man beispielsweise (wie in Abb. 12) anzugeben, um welchen Winkel sich die Schildkröte drehen soll, ist der Computer nicht fähig den Befehl auszuführen. XLogoOnline prüft das Geschriebene kontinuierlich und weist mit einer kurzen Erklärung auf den Fehler hin. So werden sich Kinder ihrer Fehler bewusst und können diese korrigieren [2].

2. Logische Fehler (im Gegensatz zu syntaktischen Fehlern) führen nicht dazu, dass ein Programm nicht ausgeführt werden kann; das Programm wird erfolgreich in ein Bild umgewandelt, allerdings entspricht dieses nicht den Erwartungen. Vor diesen Fehlern kann uns der Computer nicht schützen. Stattdessen bietet XLogoOnline einen Mechanismus, um fehlerhafte Programme Schritt für Schritt auszuführen, sodass die Fehlerstelle einfacher gefunden und der Fehler behoben werden kann (siehe Abb. 13 und 14). Dabei steuern die Kinder die Ausführung des Programms manuell und beobachten Befehl um Befehl die unmittelbare Auswirkung. Schwierige Stellen können beliebig oft durchlaufen werden mittels gezieltem Vor- und Zurückspulen.

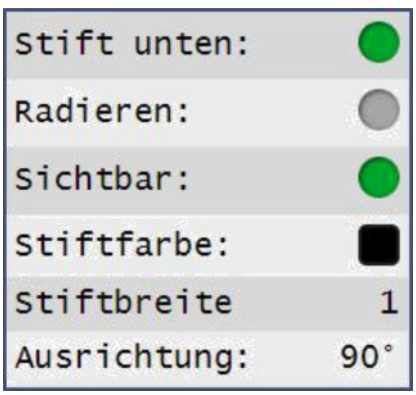


Abb. 15 Aktive Effekte der Schildkröte

In einer Tabelle (siehe Abb. 15) werden darüber hinaus diejenigen Effekte sichtbar, die nicht direkt visualisiert sind: Hat man beispielsweise vergessen die Stiftfarbe korrekt zu setzen, wird der Fehler hier ersichtlich.

Mittels textbasierter Programmierung können größere und auch komplexere Programme konstruiert werden, als dies zuvor der Fall war. Wie in natürlichen Sprachen kann sich auch Logo dynamisch entwickeln und erlaubt neue Wörter (also Befehle) zu definieren und diese anschließend zu verwenden. So können bereits geschriebene Programme einfach wiederverwendet werden, um immer größere und komplexere Konstrukte zu erzeugen. Betrachten wir dazu ein Beispiel, in welchem ein Muster aus drei Quadraten gezeichnet werden soll (siehe Abb. 16).

Abb. 16 Drei Quadrate

Das wiederkehrende Muster ist ein Quadrat der Seitenlänge 100. Wir erweitern die Sprache mit einem neuen Befehl `quadrat100`.

```
TO quadrat100
repeat 4[fd 100 rt 90]
END
```

Dieser kann anschließend verwendet werden, um das Muster zu zeichnen:

```
repeat 3[quadrat100 fd 100]
```

Wir führen neue Befehle ein, um jegliche Konstrukte übersichtlich und kurz mit einer Handvoll Befehlen zu beschreiben. Verschachtelungen dieser Art nennt man **modulares Programmieren**.

In einem nächsten Schritt erfahren Kinder, wie sie Programme schreiben, die zu mehr als nur einem Zweck eingesetzt werden können. Dazu verwenden sie das Konzept der **Parametrisierung**, welches bereits in Kapitel „Variable Distanzen und Winkel" ein erstes Mal unbewusst eingesetzt wurde. Analog zum Befehl fd, dessen Schrittlänge beliebig parametrisiert werden kann, kann auch das Programm quadrat soweit parametrisiert werden, dass dessen Seitenlänge frei bestimmt werden kann.

```
TO quadrat :laenge
repeat 4[fd :laenge rt 90]
END
```

Mit dem modularen Entwurf lernen die Kinder eine wichtige Methode, um Komplexität zu meistern. Sie lernen, wie sie Probleme in kleinere Teilprobleme zerlegen, welche sie bereits gelöst und deren Funktionalität sie getestet haben. Erst wenn die zu verwendenden Module korrekt funktionieren, fahren sie fort, um daraus ein komplexeres Muster zu erzeugen. Mittels Parametrisierung lernen sie einen Mechanismus kennen, der es ihnen erlaubt, eine Vielzahl an verschiedenen Mustern in nur einem einzigen neuen Programm zu vereinen. An dieser Stelle werden sich die Kinder bewusst, wie die verwendete Sprache der vorherigen beiden Zyklen intern funktioniert und sind fähig die verwendete Programmiersprache selbst sinnvoll zu erweitern.

Ein Schritt in die Professionalisierung

Mit dem Übertritt in die Sekundarstufe I sind Schüler kognitiv in der Lage, auch abstraktere Konzepte wie Variablen, Bedingungen und Listen erfolgreich zu meistern. Mit Python führen wir eine Programmiersprache ein, die selbst im professionellen Umfeld weit verbreitet ist und unseren didaktischen Ansprüchen genügt.

Wir entwickelten die beiden Programmierumgebungen TigerJython [1] und WebTiger-Jython (https://webtigerjython.ethz.ch, s. Abb. 17), die für Tablets, Laptops und Computer verfügbar sind. Als Lernumgebung offerieren wir den Schülern auch hier bewusst nur die Funktionalitäten, welche diese für den Unterricht benötigen. Das Kernstück der Umgebungen ist ein Mechanismus, der Schülerprogramme analysiert und im Falle von Fehlern eine automatische Meldung produziert. Diese sind in leicht verständlicher Sprache formuliert, weisen gezielt auf den Fehler hin und erlauben es den Lernenden, selbstständig ihre Programme zu korrigieren.

Eine Aufgabenstellung dieser Zielstufe lautet, Quadrate zunehmender Größe in einem Programm zu vereinen.

Die Schülerinnen wissen aus früheren Erfahrungen, wie sie ein parametrisiertes Programm *quadrat(groesse)* schreiben, um Quadrate beliebiger Größen zu zeichnen. Bei dieser Aufgabe stoßen die Jugendlichen jedoch an eine Grenze: Um das gegebene Bild zu zeichnen, müsste der Befehl 50-mal in Folge mit verschiedenen Parametern geschrieben werden.

```
quadrat(10)
quadrat(20)
quadrat(30)
...
quadrat(500)
```

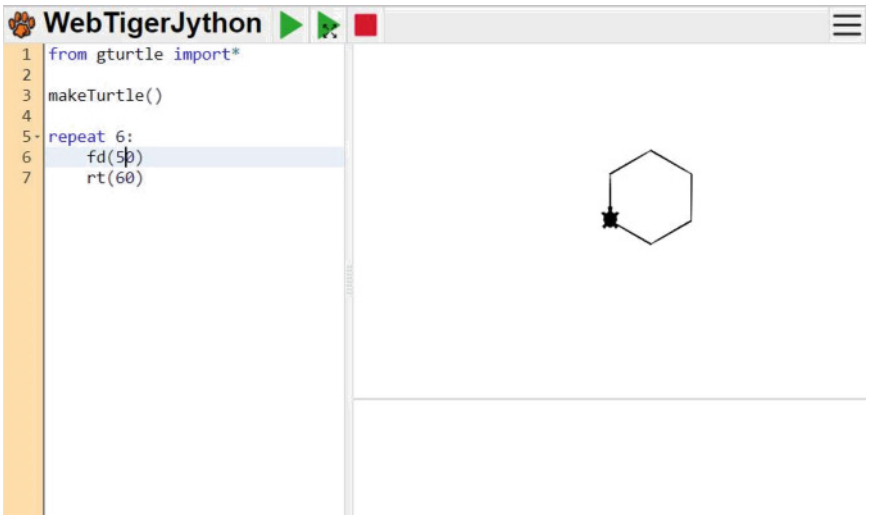


Abb. 17 WebTigerJython

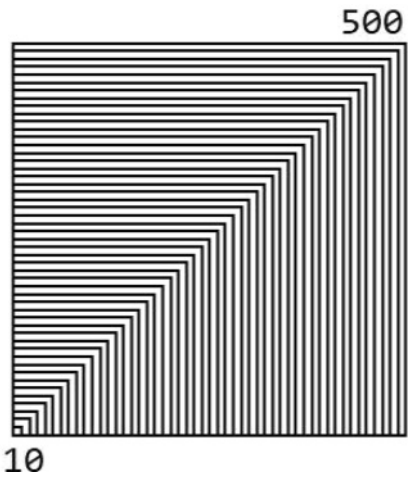


Abb. 18 Beispielaufgabe mit Variablen

An dieser Stelle offeriert sich das Konzept der Variable als hilfreiches Werkzeug für die Programmierenden: Anstatt dieselbe Zeile immer und immer wieder zu schreiben, führen sie eine Variable ein und lassen den Befehl *quadrat(groesse)* wiederholt ausführen. Die Variable „groesse“ wird in jedem Durchgang entsprechend angepasst:

```
groesse = 10
repeat 50:
    quadrat ( groesse )
    groesse = groesse + 10
```

Das Konzept einer Variable zu verstehen, ist ohne eine solide mathematische Verankerung kaum möglich. Daher empfehlen wir, dieses schwierige Konzept erst an dieser Stelle einzuführen. Dieser Schritt stellt eine wichtige Grundlage dar, um weiterführende Programmierkonzepte wie Bedingungen oder Listen angehen zu können.

Implikationen

Unser Ziel ist es, mit dem Programmieren in der Schule das kritische Denken der Schülerinnen und Schüler durch entdeckendes und selbstständiges Lernen zu fördern. Dabei wollen wir nicht eine zukünftige Generation von Informatikern erziehen, vielmehr sollen die Kinder für die digitale Zukunft vorbereitet werden, indem sie deren Entwicklung nachvollziehen. Dies ermächtigt sie dazu, selbst mitzubestimmen, wie sich die digitale Welt weiterentwickeln soll. Sie lernen an komplexe Aufgabenstellungen heranzugehen und Lösungsstrategien zu finden. Das Arbeiten mit einer Programmiersprache und das Erweitern der Sprache mit eigenen Befehlen, bietet ein universelles Werkzeug, um zukünftige Probleme anzugehen. Auf diese Weise werden die Lernenden zu Problemlösern, Schaffern und Erfindern und konsumieren nicht nur die Erfindungen anderer.

Literatur

1. Arnold J, Kohn T, Plüss A. http://www.tigerjython.ch, letzter Zugriff: 20.1.2019
2. Forster M et al. (2018) Autonomous recovery from programming errors made by primary school children. International Conference on Informatics in Schools: Situation, Evolution, and Perspectives. Springer, Cham
3. Hromkovič J (2017) Einfach Informatik 7/9 – Programmieren. Klett und Balmer AG, Zug
4. Hromkovič J (2017) Einfach Informatik 7/9 – Programmieren (Begleitband). Klett und Balmer AG, Zug
5. Hromkovič J, Lacher R (2017) The computer science way of thinking in human history and consequences for the design of computer science curricula. In: Proc. of ISSEP 2017. LNCS 10696:3–11

6. Hromkovič J, Kohn T, Komm D, Serafini G (2016) Combining the power of python with the simplicity of logo for a sustainable computer science education. In: 9th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP 2016), Münster, Germany, October 13–15, 2016. Springer, Cham, pp 155–166
7. Hromkovič J, Serafini G, Staub J (2017) XLogoOnline: a single-page, browser-based programming environment for schools aiming at reducing cognitive load on pupils. In: International Conference on Informatics in Schools: Situation, Evolution, and Perspectives. Springer, Cham
8. krgarden.ca und tts-international.com, letzter Zugriff: 22.2.2019