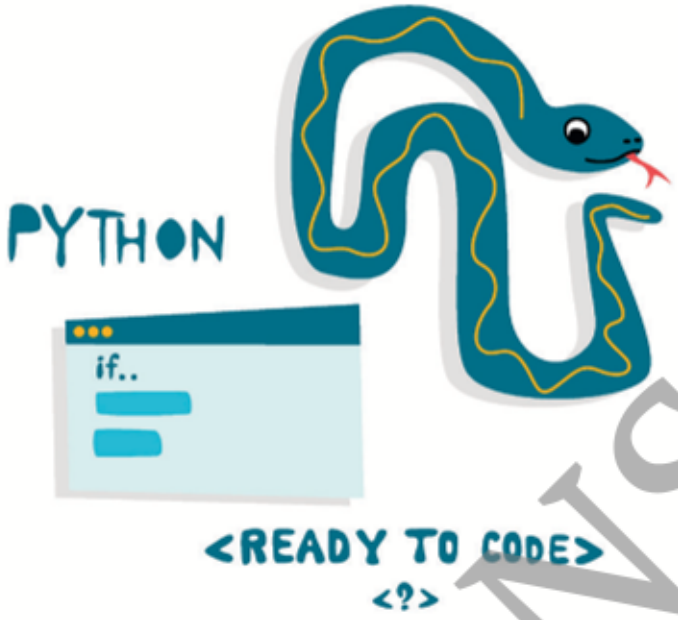
Inhaltsfeld Algorithmen

Programmierunterricht mit dem PRIMM-Lehrkonzept strukturieren – Umsetzungsbeispiel mit *Python*

Ein Beitrag von Jonathan Pfeiffer



Gerade für Programmieranfängerinnen und -anfänger ist es oftmals nicht einfach motiviert und erfolgreich am Ball zu bleiben. Ein in der praktischen Unterrichtsumgebung bewährter Ansatz ist das sogenannte PRIMM-Lehrkonzept zur Strukturierung Ihres Programmierunterrichts. Dieser Ansatz setzt das Quellcode lesen vor das Quellcode schreiben. Kollaboratives Lernen wird in den Fokus gesetzt, damit die Lernenden über die Programme und Konzepte sprechen. Dabei wird auf eine Reduzierung der kognitiven Belastung der Programmieranfängerinnen und -anfänger geachtet, indem sich die Lernenden schrittweise und allmählich ein Programm zu eigen machen. Dieser Beitrag stellt Ihnen das PRIMM-Konzept mit kleinen Praxisbeispielen innerhalb der Programmierumgebung *Python* für die Umsetzung im Unterricht vor.

METHODIK & DIDAKTIK

Klassenstufe:	7–10 + Sek. II
Thematische Bereiche:	Algorithmen, Programmieren, Programmierunterricht, Umsetzung an Beispielen in *Python*
Kompetenzbereiche:	Implementieren, Kooperieren und Kommunizieren
Methodischer Ansatz:	*Use-Modify-Create*-Konzept, PRIMM-Konzept
Sozialform:	Partnerarbeit

Definition PRIMM

In der nationalen und internationalen methodisch-didaktischen Fachliteratur finden sich verschiedene Frameworks und didaktische Ansätze, die sich mit der Entwicklung der Schülerinnen und Schüler beim Programmieren Lernen auseinandersetzen. In diesem Beitrag soll PRIMM als ein sinnvoller Ansatz zur guten Strukturierung des Informatikunterrichts, insbesondere des Programmierunterrichts gerade für Programmieranfängerinnen und -anfänger, vorgestellt werden.

Die Abkürzung **PRIMM** steht für:

- P = *predict* (vorhersagen)
- R = *run* (ausführen)
- I = *investigate* (untersuchen)
- M = *modify* (anpassen)
- M = *make* (machen)

Use-Modify-Create-Lehrkonzept als Grundlage

Der PRIMM-Ansatz bezieht sich auf das ***Use-Modify-Create*-Lehrkonzept** (Lee et al. 2011, 35). Dieses gliedert sich in die drei Phasen *Use (Nutzen)*, *Modify* (Modifizieren) und *Create (Gestalten)*.

Use-Phase

In der *Use*-Phase, die im PRIMM-Ansatz *predict*, *run* und *investigate* entspricht, sind die Schülerinnen und Schüler Konsumenten der von jemand anders erstellten „Kreationen". D.h. die Programmieranfängerinnen und -anfänger arbeiten in dieser ersten Phase zunächst mit bereits bestehenden und fertigen Programmen, nutzen also Programme, an deren Erstellungsprozess sie unbeteiligt sind („nicht meins"). Es kann sich dabei auch um die Anwendung bestehender Computermodelle oder ein fertig erstelltes Computerspiel handeln. Das entlastet die Lernenden in vielerlei Hinsicht. Da es „nicht meins" ist, besteht keine emotionale Beziehung zu diesen Programmen. Die kognitive Belastung beim Programmieren (Problem verstehen, Algorithmus entwickeln, Fehlermeldungen interpretieren, Fehler suchen usw.) wird reduziert, weil es zunächst einmal nur um das Nutzen, Ausprobieren, Analysieren und Verstehen geht. Zudem kann der Fokus auf ausgewählte Aspekte gelegt werden, z. B. Syntax, Reihenfolge, häufige Fehlerquellen usw. Haben die Lernenden das Programm ausreichend z. B. mithilfe einer Syntax-Checkliste, eines Fehlerprotokolls oder geeigneten Analyse-Fragen (Was ist neu? Was bewirkt der neue Bestandteil?) untersucht und das zugrundeliegende Konzept (zumindest teilweise) verstanden, dann können (und wollen) sie das Programm verändern und modifizieren. Meist entwickeln die Lernenden selbst bei der Anwendung des bereits bestehenden Programms die Motivation Veränderungen vorzunehmen. Dies können zunächst kleine Änderungen, wie z. B. bei einem Videospiel die eingestellte Hintergrundfarbe sein, was sich aber mit zunehmenden Programmierkenntnissen mehr und mehr ausbauen kann. Wichtig ist, dass die Herausforderungen nur allmählich gesteigert werden sollten. So kann verhindert werden, dass die Schülerinnen und Schüler vor einer Aufgabe Angst haben müssen und sie fühlen sich den nach und nach kommenden Herausforderungen stets gewachsen.

Umsetzungsbeispiel des PRIMM-Ansatzes im Unterricht

Im Folgenden wird Ihnen der PRIMM-Ansatz an einem klassischen Unterrichtsthema im Programmierunterricht veranschaulicht. Sie erhalten dabei für das erläuterte PRIMM-Lehrkonzept an einem Umsetzungsbeispiel für **Alternativen (if-else-Verzweigung)** in *Python* geeignete Unterrichtsmaterialien.

Vorwissen für dieses Thema

Die Lernenden können bereits…

- Anweisungen sequentieller Abläufe in die richtige Reihenfolge bringen.
- eine bedingte Anweisung (if-Anweisung) formulieren.
- eine Bedingung einer bedingten Anweisung selbstständig bestimmen.
- eine Ausgabe mittels `print()` schreiben.
- eine Eingabe mittels `input()` einlesen und einer Variable zuweisen.
- eine Eingabe mit `int()` in eine Ganzzahl umwandeln.
- einen Wert einer Variable zuweisen.
- ein *Python*-Programm in einer Entwicklungsumgebung speichern und ausführen.

Methodisch-didaktische Überlegungen zu den Materialien

create bzw. make

Gemäß des vorgestellten PRIMM-Ansatzes und *Use-Modify-Create*-Lehrkonzeptes beginnen die Materialien in der *Use*-Phase (PRIMM: *predict – run – investigate*). Hierfür startet **M 1** mit einem vorgegebenen „fremden" Programmiercode zu einem Fahrkartenautomat. Die Schülerinnen und Schüler nutzen den Code zunächst im Sinne von „nicht meins", indem sie diesen in eine *Python*-Entwicklungsumgebung kopieren und systematisch unter Verwendung einer Syntax-Checkliste auf Syntax-Fehler hin untersuchen. Anschließend testen sie das Programm, indem sie es ausführen. Als Abschluss dieser Phase erläutern sich die Lernenden gegenseitig den Programmablauf und korrigieren sich bei Bedarf.

Hinweis: Als *Python*-Entwicklungsumgebung bietet sich beispielsweise *Thonny* an.

modify

M 2 wird für die anschließende *Modify*-Phase (PRIMM: *modify*) eingesetzt. Hier modifizieren die Schülerinnen und Schüler das vorliegende Programm z. B. um veränderte Fahrtkartenpreise. Diese Phase wird in **M 3** an einem weiteren Programmiercode zu einem Kinoticketautomaten gefestigt, indem die einzelnen Schritte hieran nochmals wiederholt werden. Da die Schülerinnen und Schüler hier schon einzelne wesentliche Teile, nämlich die if-else-Elemente, des Codes selbst schreiben, werden sie langsam von „nicht meins" zu „meins" herangeführt.

Use bzw. predict – run – investigate

In **M 4** gehen die Lernenden schließlich vollständig in die *Create*-Phase (PRIMM: *make*) über, indem sie am Beispiel eines Portorechners auf Grundlage der erlernten Kenntnisse ein eigenes Programm mit if-else-Verzweigung in *Python* schreiben.

Dieses Konzept lässt sich problemlos auf andere Informatikkonzepte und Programmierprobleme, wie zum Beispiel Schleifen, übertragen.

Merke

Die Alternative besteht aus einer Bedingung, deren Wahrheitswert überprüft wird.
Je nach Ergebnis dieser Prüfung wird einer von zwei Anweisungsblöcken ausgeführt:

- Ist die Bedingung erfüllt (Wahr/True), wird der if-Block ausgeführt.
- Ist die Bedingung nicht erfüllt (Falsch/False), wird der else-Block ausgeführt.

```
1  if bedingung(en):
2     #if-Block
3     anweisung(en)
4  else:
5    #else-Block
6     anweisung(en)
```