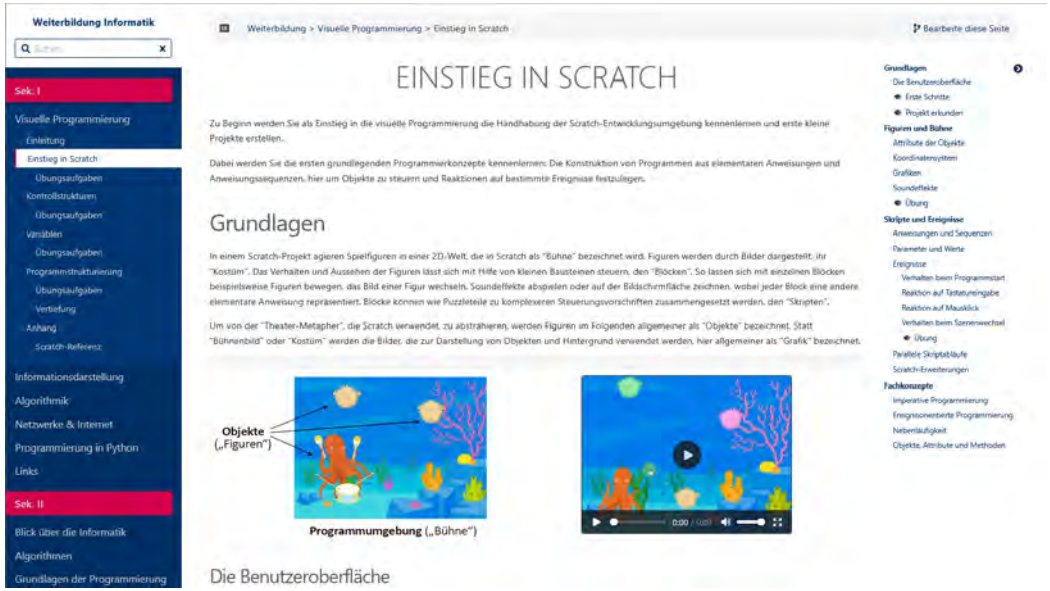
Code verstehen kurz gefasst

… mit PRIMM, Use-Modify-Create, Parsons Problems, Tracing, Live-Coding, Blockmodell & KI

Peer Stechert

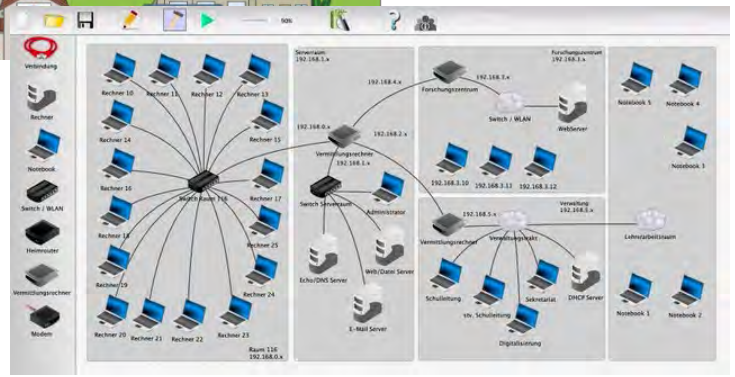
peer.stechert@mnu.de

Über mich



Lernsoftware
Pattern Park

Lernsoftware
Filius



Weiterbildung Informatik:
https://sekii.informatik-sh.de

Agenda

- Müssen wir noch Programmieren lernen?
- Programmieren in den neuen GI-Bildungsstandards Sek. I
- Cognitive Load Theorie – Warum Programmieren lernen so schwierig ist
- Prinzip 1: Code lesen vor Code schreiben
- Prinzip 2: Unterricht strukturieren
- Prinzip 3: Programmverstehen systematisieren
- Ausblick: Neue Prüfungsformate im Fach Informatik

Müssen wir noch Programmieren lernen?

“English is the new most important programming language”

– Karpathy 2023

Ja, aber anders …

Analyse, Planung und Verifikation werden wichtiger!

SIGCSE TS 2026 - Saturday Keynote: "CS and SE Education, post-AI" by Titus Winters
https://www.youtube.com/watch?v=6Fyjf3gFjUk

Die neuen GI-Bildungsstandards Sek. I

- In Jahrgangsstufe 5-6 Empfehlung einer visuellen Programmiersprache, danach visuell oder textbasiert.

https://informatikstandards.de/

4.1 Modellieren und Implementieren (MI)

Modellieren und Implementieren sind zentrale Elemente des informatischen Problemlöseprozesses, bei dem ein Problem analysiert, ein Modell entworfen und implementiert wird. Die einzelnen Schritte und der gesamte Prozess werden reflektiert.

5.2 Algorithmen (AL)

Jahrgangsstufen 5 bis 6

Die Schülerinnen und Schüler ...

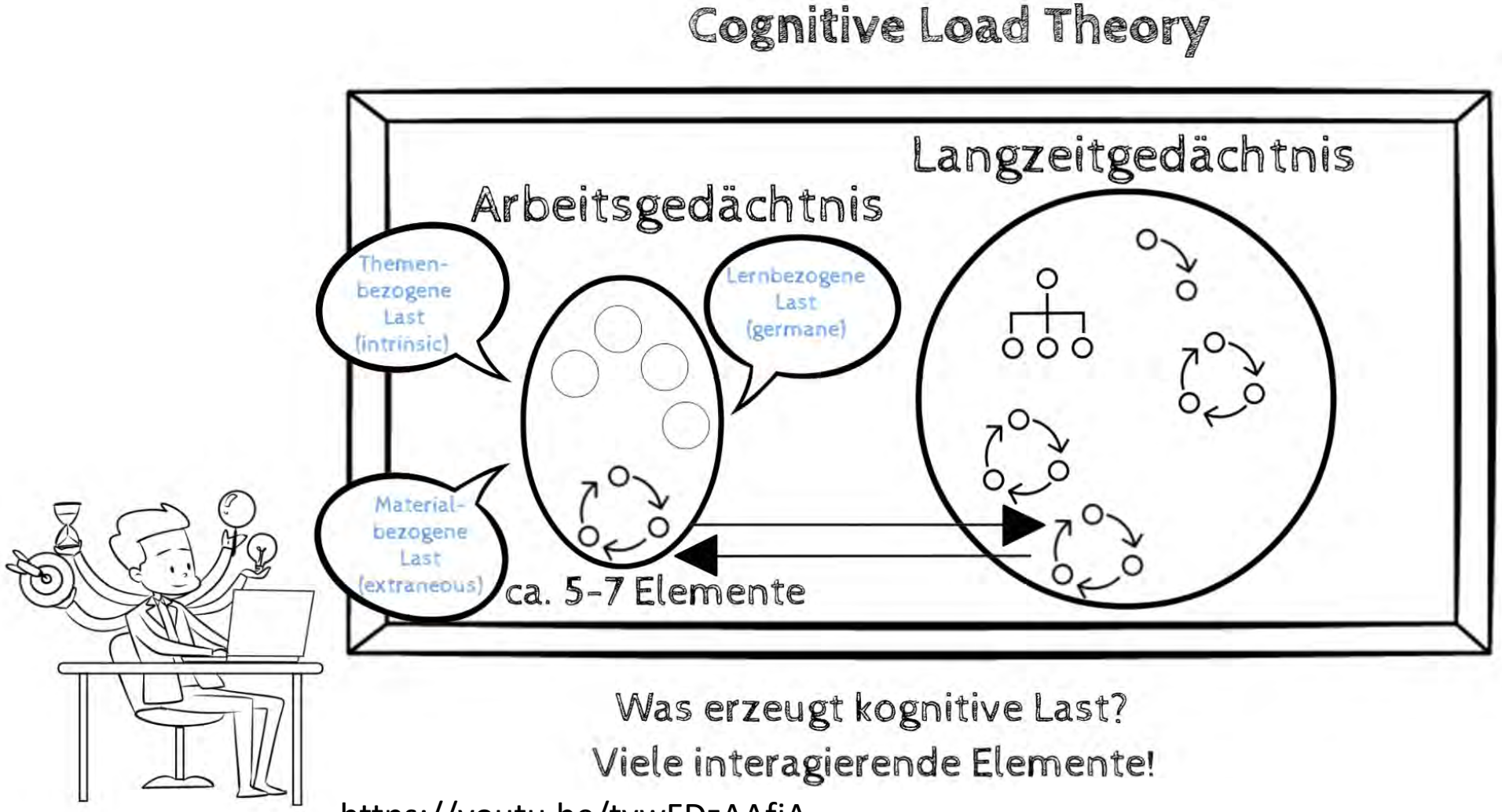
- AL-1 beschreiben Algorithmen, die in einer altersangemessenen Form dargestellt sind.
- AL-2 modellieren Algorithmen unter Verwendung algorithmischer Grundbausteine.
- AL-3 implementieren Algorithmen in einer visuellen Programmiersprache.
- AL-4 testen, ob ein Programm eine gegebene Aufgabenstellung löst, und beheben Fehler.

Jahrgangsstufen 7 bis 10

Die Schülerinnen und Schüler ...

- AL-5 vollziehen gegebene Algorithmen nach und analysieren sie, u. a. hinsichtlich Funktion oder Aufwand.
- AL-6 analysieren Problemstellungen, zerlegen diese in Teilprobleme und modellieren Algorithmen zu deren Lösung.
- AL-7 implementieren Algorithmen in einer visuellen oder textbasierten Programmiersprache.
- AL-8 testen ihre Programme systematisch, finden Ursachen von Fehlern und beheben diese.
- AL-9 beschreiben an Beispielen Ansätze der Künstlichen Intelligenz zur Lösung von Problemen.

Vorüberlegung: Warum Programmieren lernen so schwierig ist



https://youtu.be/tywFDzAAfjA

Programmieren und kognitive Last

Programmieren erzeugt eine hohe kognitive Last, die das gleichzeitige Lernen wichtiger Konzepte verhindert.


```
int x = 10; int y = 5; int z = 3;
if ( x<= y) {
  if (y > 5) {
    z = 5;
  } else {
    z = 6;
  }
}
```

- Datentypen
- Variable (Dekl. & Initial.)
- Zuweisung
- Bedingte Anweisung
- Boolescher Ausdruck
- Vergleiche
- Blöcke
- Bezeichner

(Beispiel nach Mühling, 2019)

https://youtu.be/tywFDzAAfjA

Programmieren und kognitive Last

Programmieren erzeugt eine hohe kognitive Last, die das gleichzeitige Lernen wichtiger Konzepte verhindert.


```
int x = 10; int y = 5; int z = 3;
if ( x<= y) {
  if (y > 5) {
    z = 5;
  } else {
    z = 6;
  }
}
```

„Chunking"

(Stechert, 2009)
(PatternPark, 2021)

- Datentypen
- Variable (Dekl. & Initial.)
- Zuweisung
- Bedingte Anweisung
- Boolescher Ausdruck
- Vergleiche
- Blöcke
- Bezeichner

(Beispiel nach Mühling, 2019)

Programmieren und kognitive Last

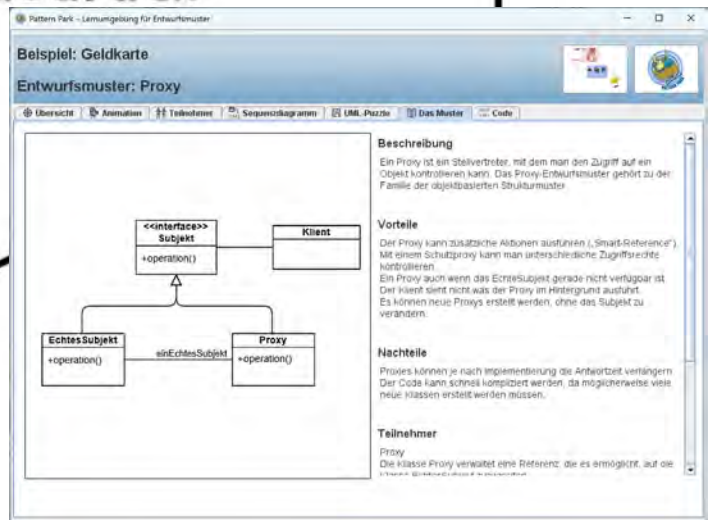
Programmieren erzeugt eine hohe kognitive Last, die das gleichzeitige Lernen wichtiger Konzepte verhindert.


```
int x = 10; int y = 5; int z = 3;
if ( x<= y) {
  if (y > 5) {
    z = 5;
  } else {
    z = 6;
  }
}
```

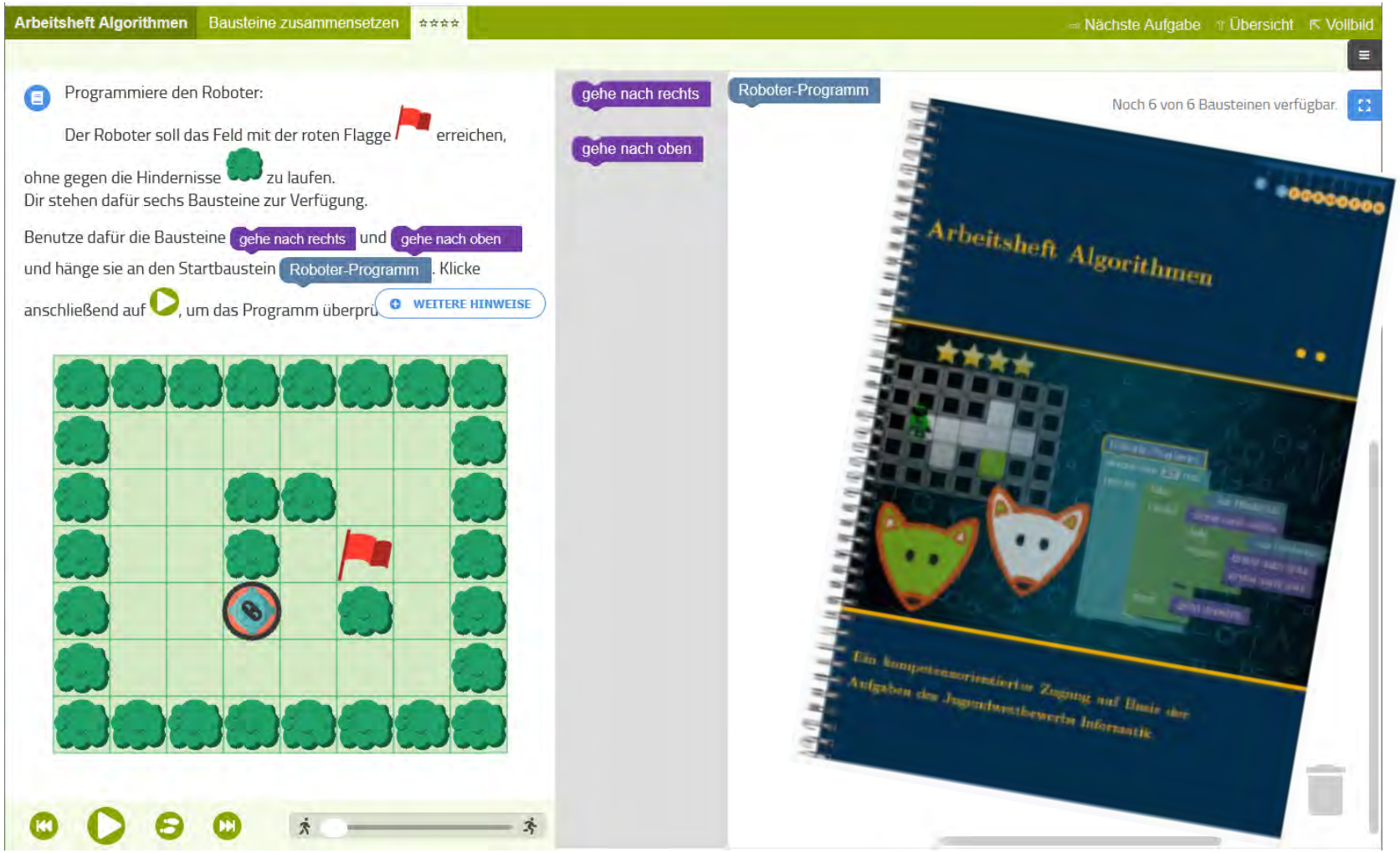
- Datentypen
- Variable (Dekl. & Initial.)
- Zuweisung
- Bedingte Anweisung
- Boolescher Ausdruck
- Vergleiche
- Blöcke
- Bezeichner

„Chunking"

(Stechert, 2009)
(PatternPark, 2021)



Werkzeugbezogene Last minimieren: Blöcke

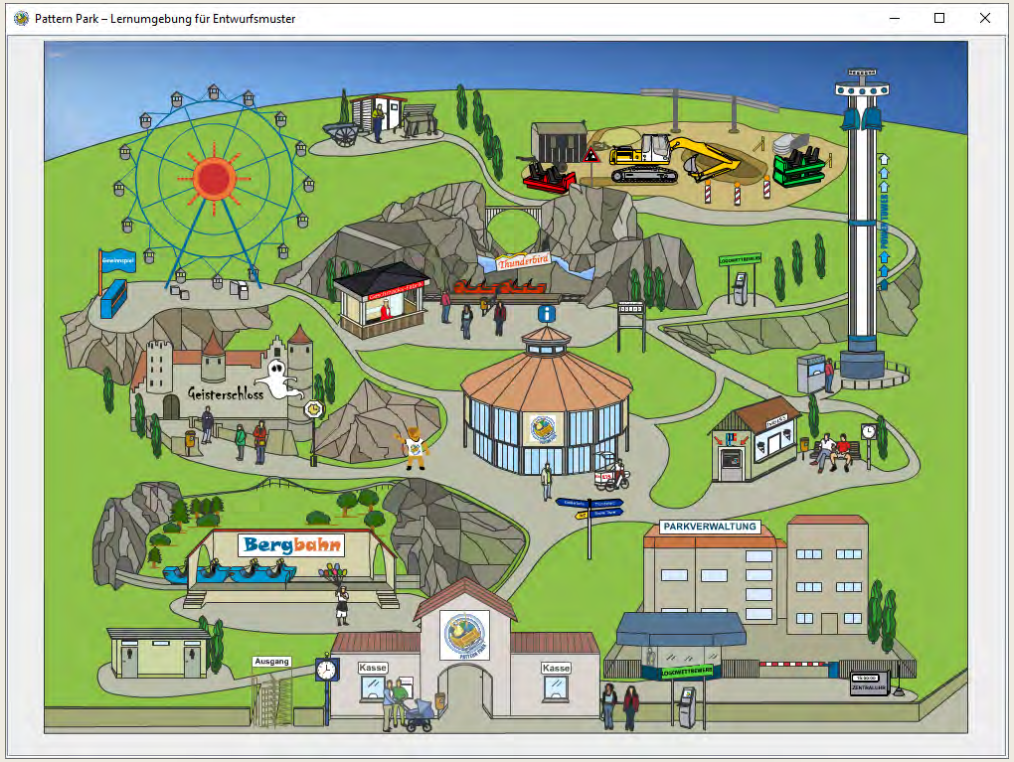


https://jwinf.de/contest/training

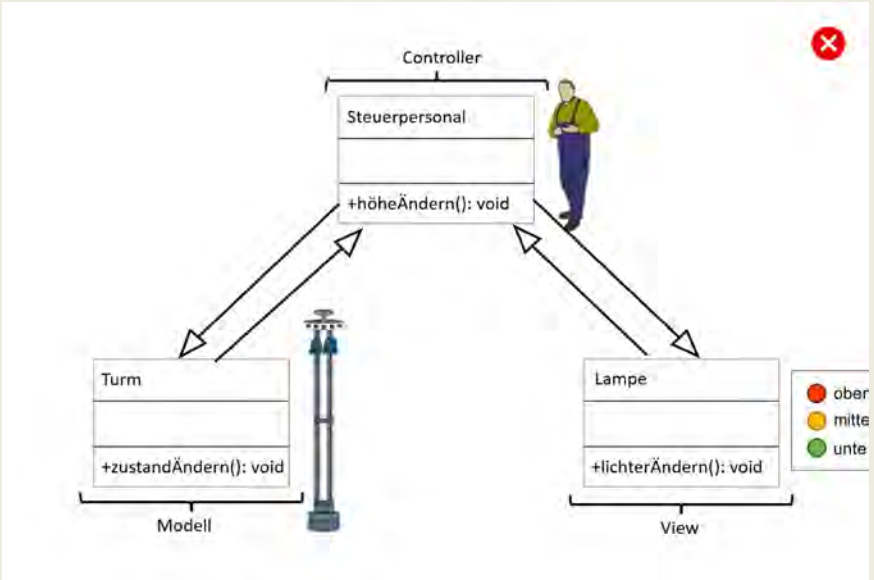
Werkzeugbezogene Last minimieren: Frames

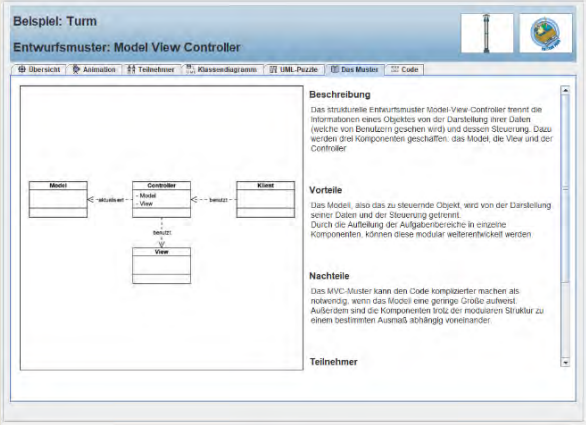
https://strype.org/

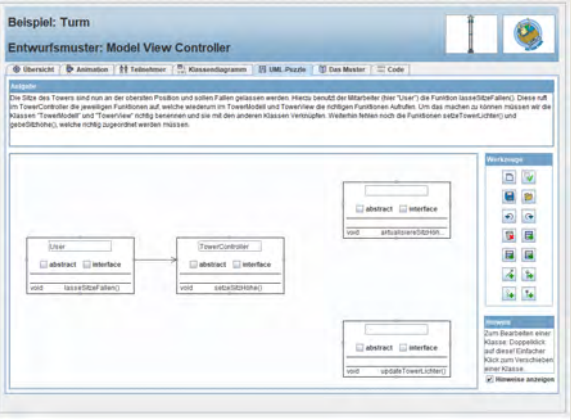
Entwurfsmuster zur Schema-Bildung



https://gitlab.com/patternpark/pattern-park-app/-/releases



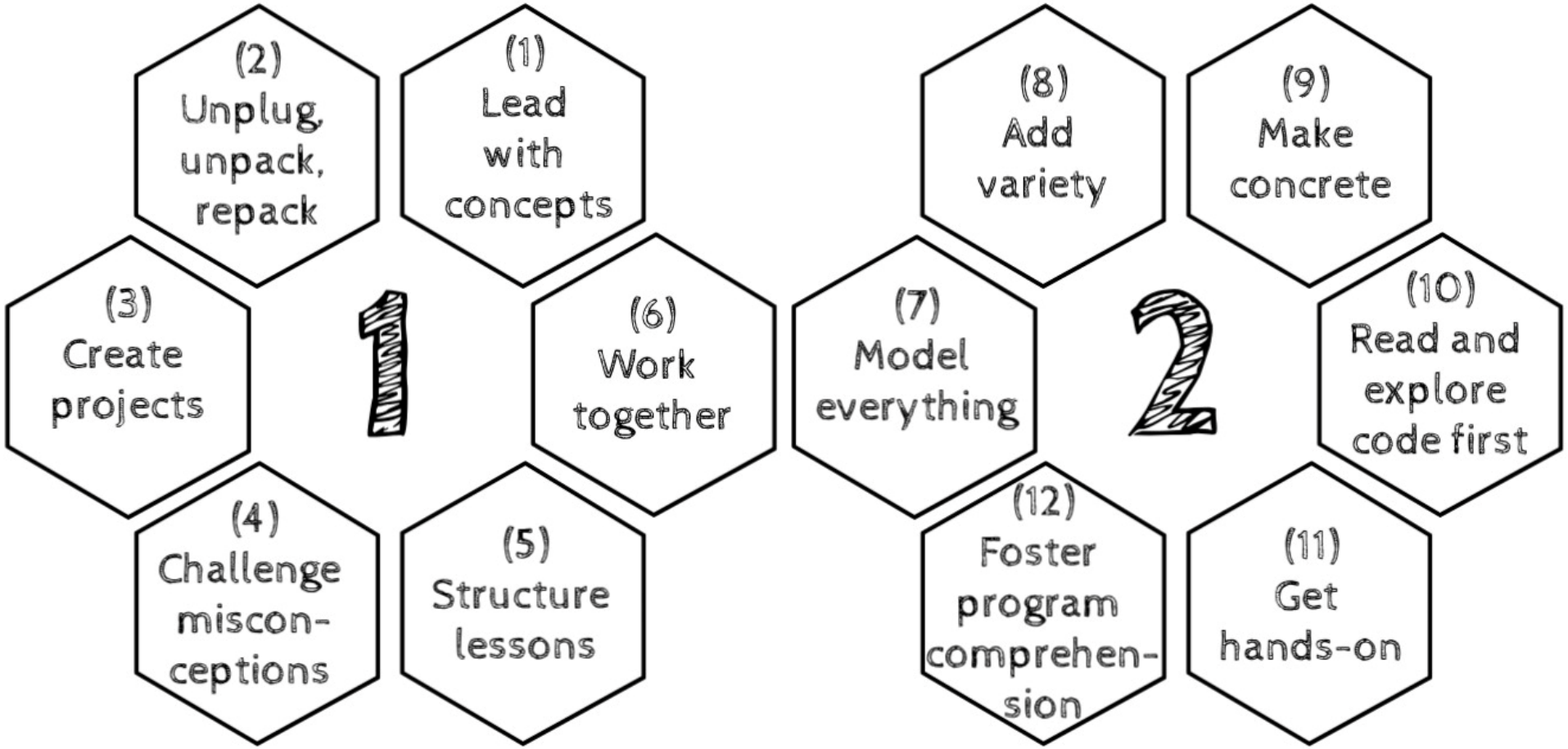


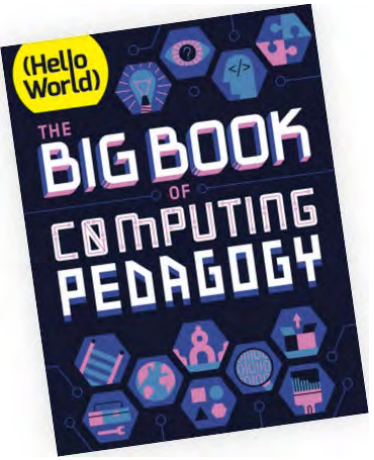


Zwischenfazit

Guter Unterricht reduziert Belastung.

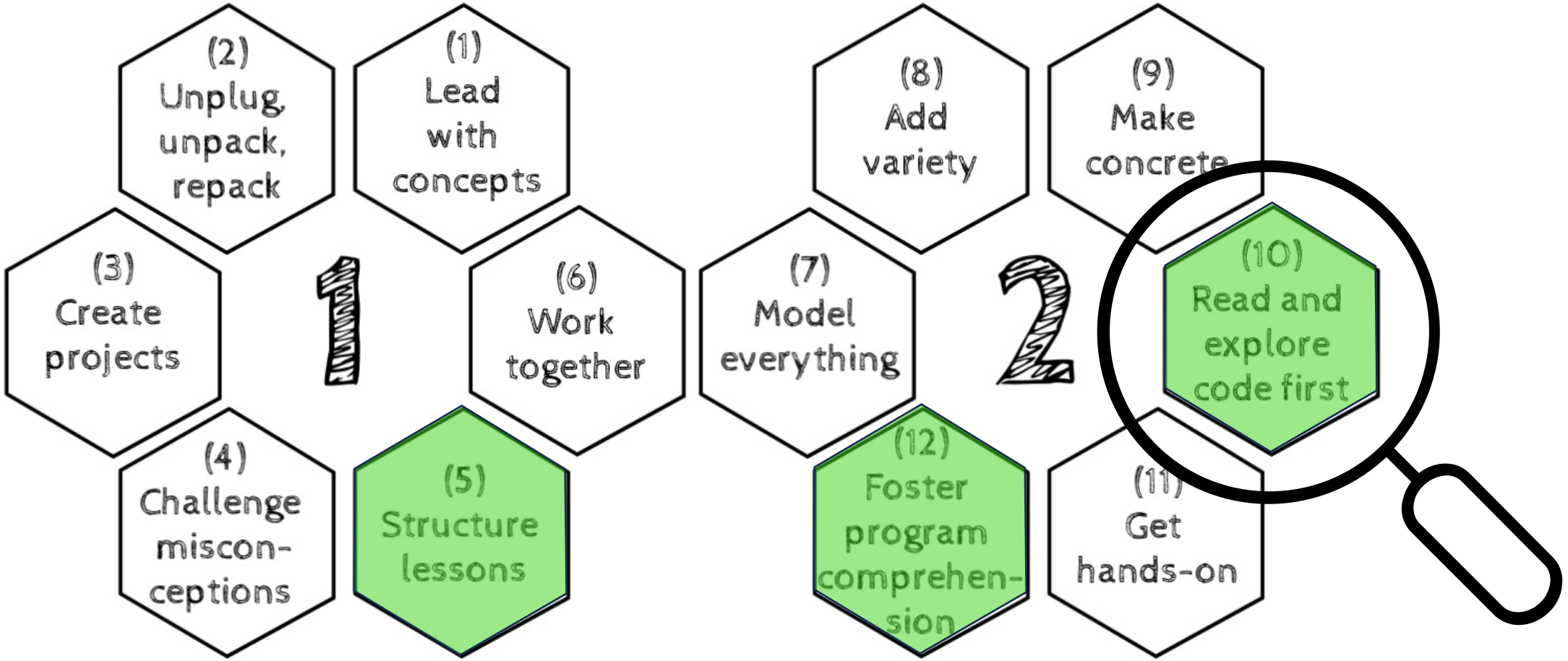
12 Fachdidaktische Prinzipien des NCCE

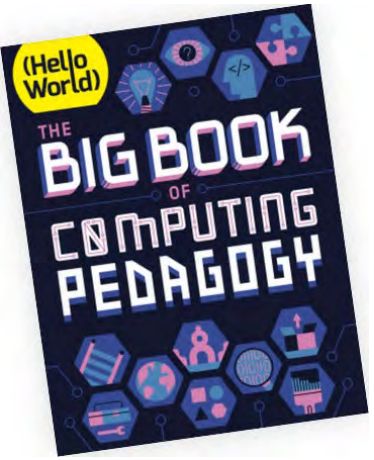




https://youtu.be/IWRGHdgntog

12 Fachdidaktische Prinzipien des NCCE





https://youtu.be/IWRGHdgntog

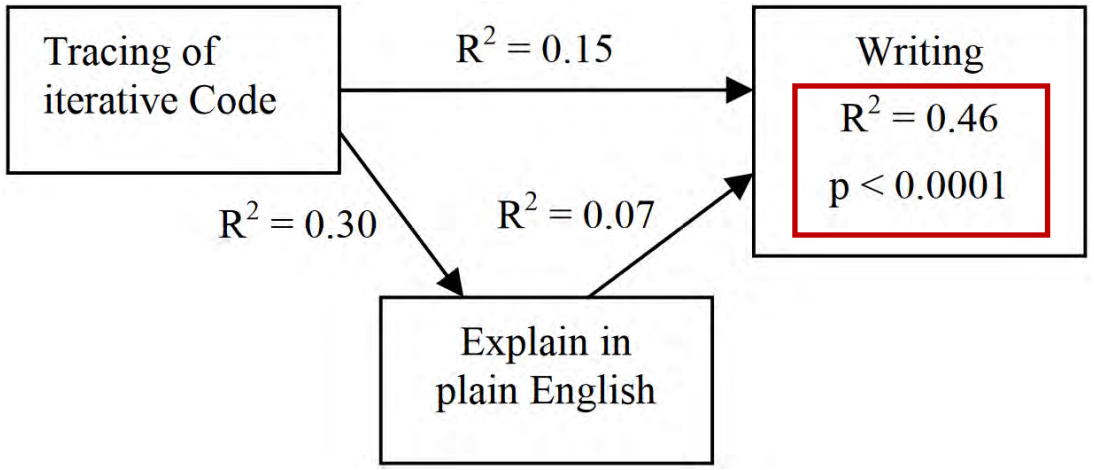
Kernthese

Code lesen ist zentral.

Prinzip 1: Code lesen vor schreiben

Typische Aufgabenformate:

- 🗨️ Programmcode erklären
- 🔍 Tracing
- 🧩 Parsons Puzzle



„Erklären“ UND „Tracing“ gemeinsam erklären 46% der Varianz beim Code schreiben!

→ starke Prädiktoren für Schreibkompetenz

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming.

Programmcode erklären

- Erkläre den Code deinem Nachbarn / deiner Nachbarin.

https://sekii.informatik-sh.de/chapter1/part3/

Tracing

Ein einfacher „Python-Computer"

(in Anlehnung an Mühling, 2020)

```
1.  jahr = int(input('Jahreszahl eingeben: '))
2.  if jahr % 400 == 0:
3.      print('Schaltjahr')
4.  elif jahr % 100 == 0:
5.      print('Kein Schaltjahr')
6.  elif jahr % 4 == 0:
7.      print('Schaltjahr')
8.  else:
9.      print('Kein Schaltjahr')
```

Aktuelle Zeile:

1

Variablen-Speicher:

Variablen-Name:	Wert:
jahr	

Bedingung:

Konsole:

```
>>>
Jahreszahl eingeben:
2024
```

Tracing

Ein einfacher „Python-Computer"

(in Anlehnung an Mühling, 2020)

```
1.  jahr = int(input('Jahreszahl eingeben: '))
2.  if jahr % 400 == 0:
3.      print('Schaltjahr')
4.  elif jahr % 100 == 0:
5.      print('Kein Schaltjahr')
6.  elif jahr % 4 == 0:
7.      print('Schaltjahr')
8.  else:
9.      print('Kein Schaltjahr')
```

Aktuelle Zeile:

2

Variablen-Speicher:

Variablen-Name:	Wert:
jahr	2024

Bedingung:

```
jahr % 400 == 0
false
```

Konsole:

```
>>>
Jahreszahl eingeben:
2024
```

Tracing

Ein einfacher „Python-Computer"

(in Anlehnung an Mühling, 2020)

```
1.  jahr = int(input('Jahreszahl eingeben: '))
2.  if jahr % 400 == 0:
3.      print('Schaltjahr')
4.  elif jahr % 100 == 0:
5.      print('Kein Schaltjahr')
6.  elif jahr % 4 == 0:
7.      print('Schaltjahr')
8.  else:
9.      print('Kein Schaltjahr')
```

Aktuelle Zeile:

4

Variablen-Speicher:

Variablen-Name:	Wert:
jahr	2024

Bedingung:

```
jahr % 100 == 0
false
```

Konsole:

```
>>>
Jahreszahl eingeben:
2024
```

Tracing

Ein einfacher „Python-Computer"

(in Anlehnung an Mühling, 2020)

```
1.  jahr = int(input('Jahreszahl eingeben: '))
2.  if jahr % 400 == 0:
3.      print('Schaltjahr')
4.  elif jahr % 100 == 0:
5.      print('Kein Schaltjahr')
6.  elif jahr % 4 == 0:
7.      print('Schaltjahr')
8.  else:
9.      print('Kein Schaltjahr')
```

Aktuelle Zeile:

6

Variablen-Speicher:

Variablen-Name:	Wert:
jahr	2024

Bedingung:

```
jahr % 4 == 0
true
```

Konsole:

```
>>>
Jahreszahl eingeben:
2024
```

Tracing

Ein einfacher „Python-Computer"

(in Anlehnung an Mühling, 2020)

```
1.  jahr = int(input('Jahreszahl eingeben: '))
2.  if jahr % 400 == 0:
3.      print('Schaltjahr')
4.  elif jahr % 100 == 0:
5.      print('Kein Schaltjahr')
6.  elif jahr % 4 == 0:
7.      print('Schaltjahr')
8.  else:
9.      print('Kein Schaltjahr')
```

Aktuelle Zeile:

7

Variablen-Speicher:

Variablen-Name:	Wert:
jahr	2024

Bedingung:

```
jahr % 4 == 0
true
```

Konsole:

```
>>>
Jahreszahl eingeben:
2024
Schaltjahr
```

„Notional Machine“

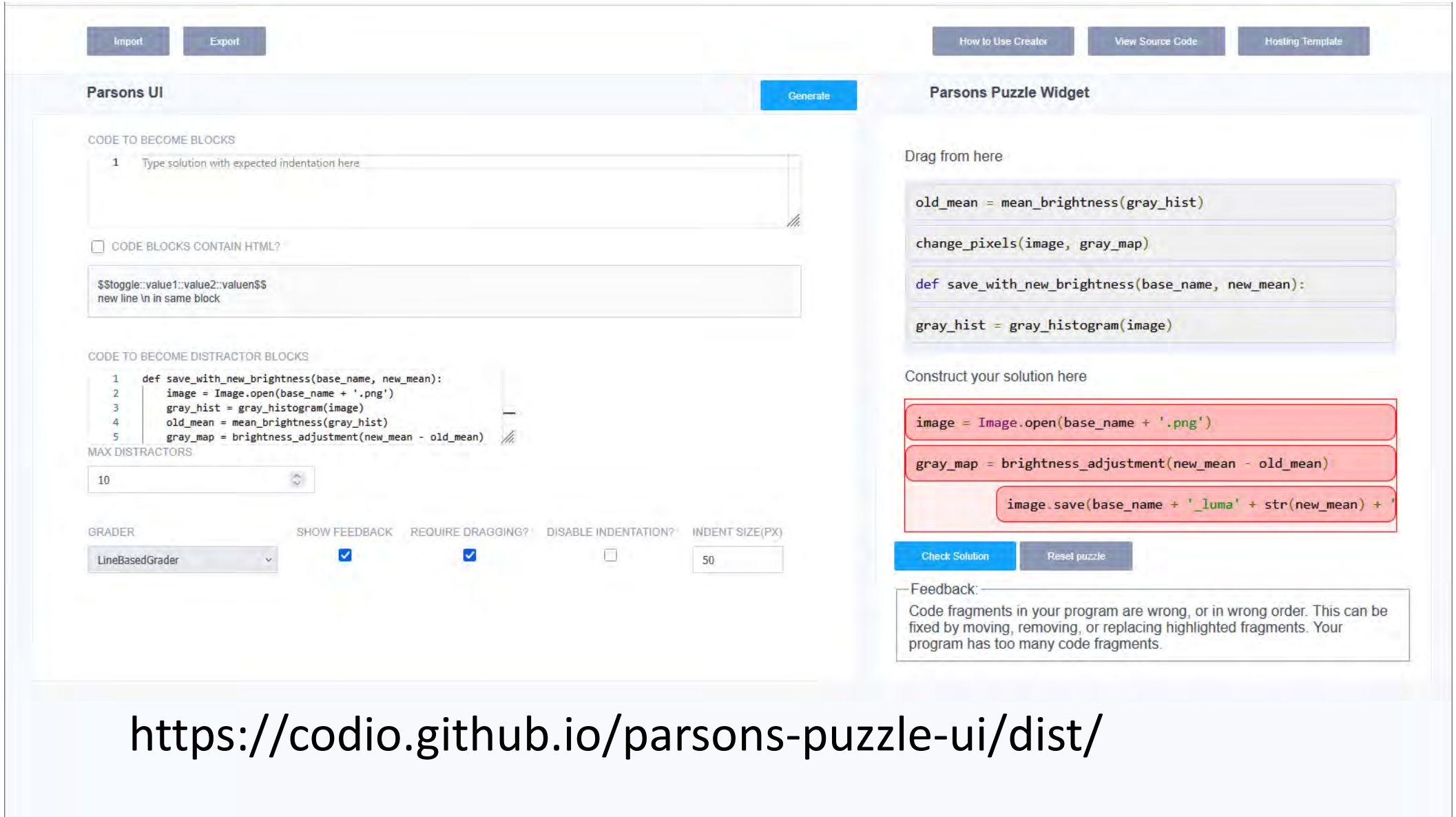
Parsons Puzzles

- Fragmentierter Programmcode
- Ziel:
  - Codefragmente in die richtige Reihenfolge
  - und Einrückungstiefe bringen
- Guter Prädiktor für die Programmierfähigkeit
  (Denny et al., 2008)

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems.

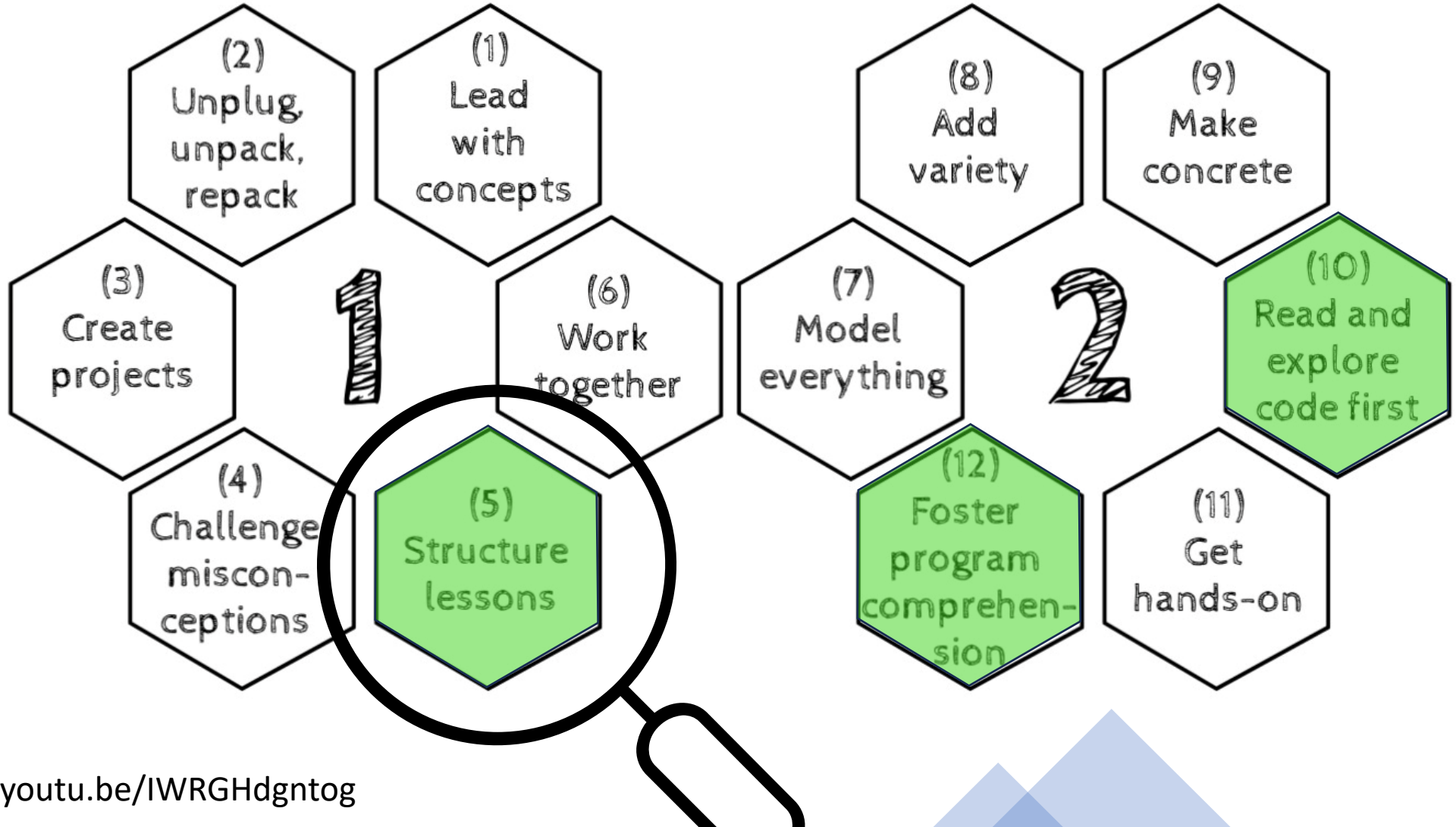
https://peerstechert.github.io/parsons-puzzles/

Parsons-Problems-Generieren



https://codio.github.io/parsons-puzzle-ui/dist/

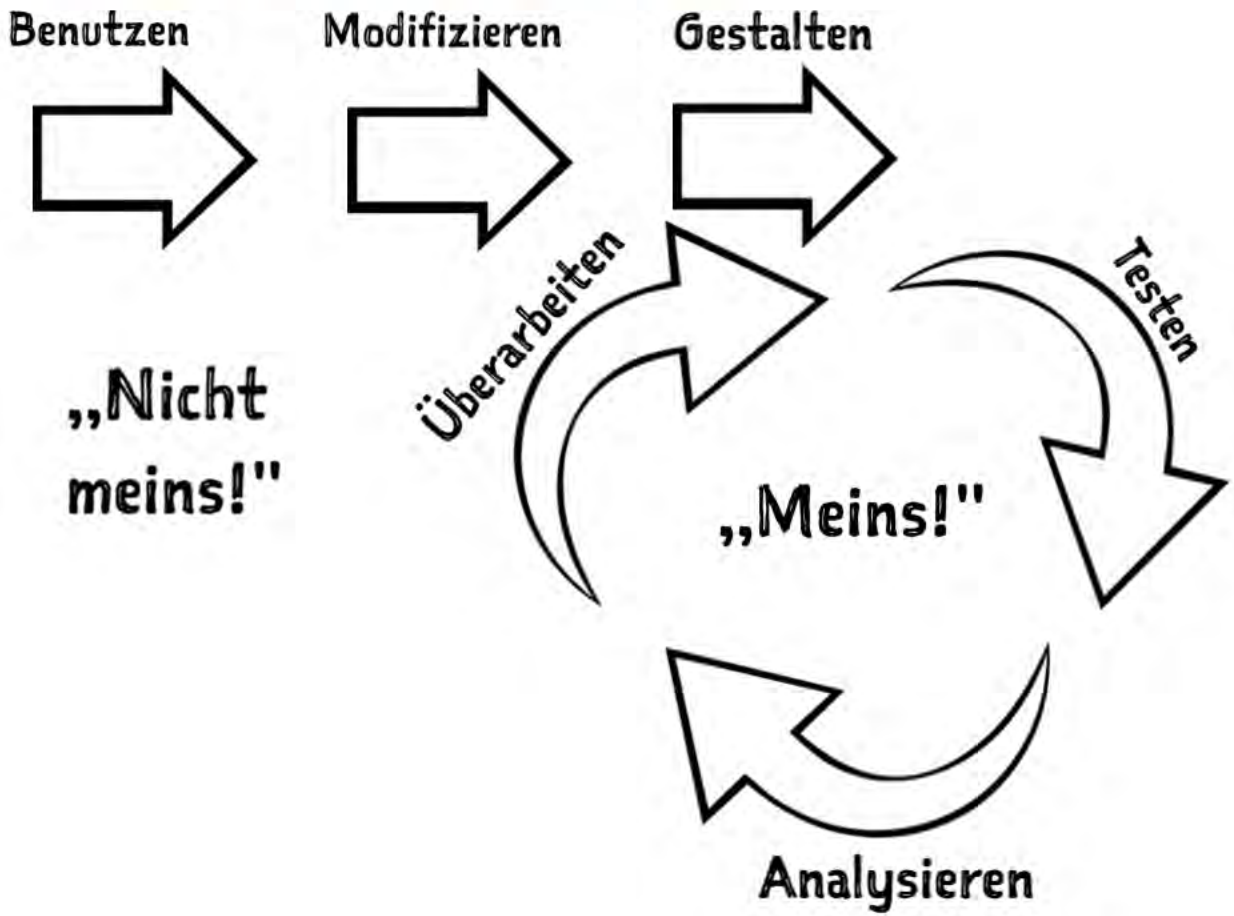
Prinzip 2: Strukturierte Lernpfade



https://youtu.be/IWRGHdgntog

Scaffolding-Strategien für das Programmierenlernen in der Schule

Use-Modify-Create



https://youtu.be/lkpBAbEQ6g0

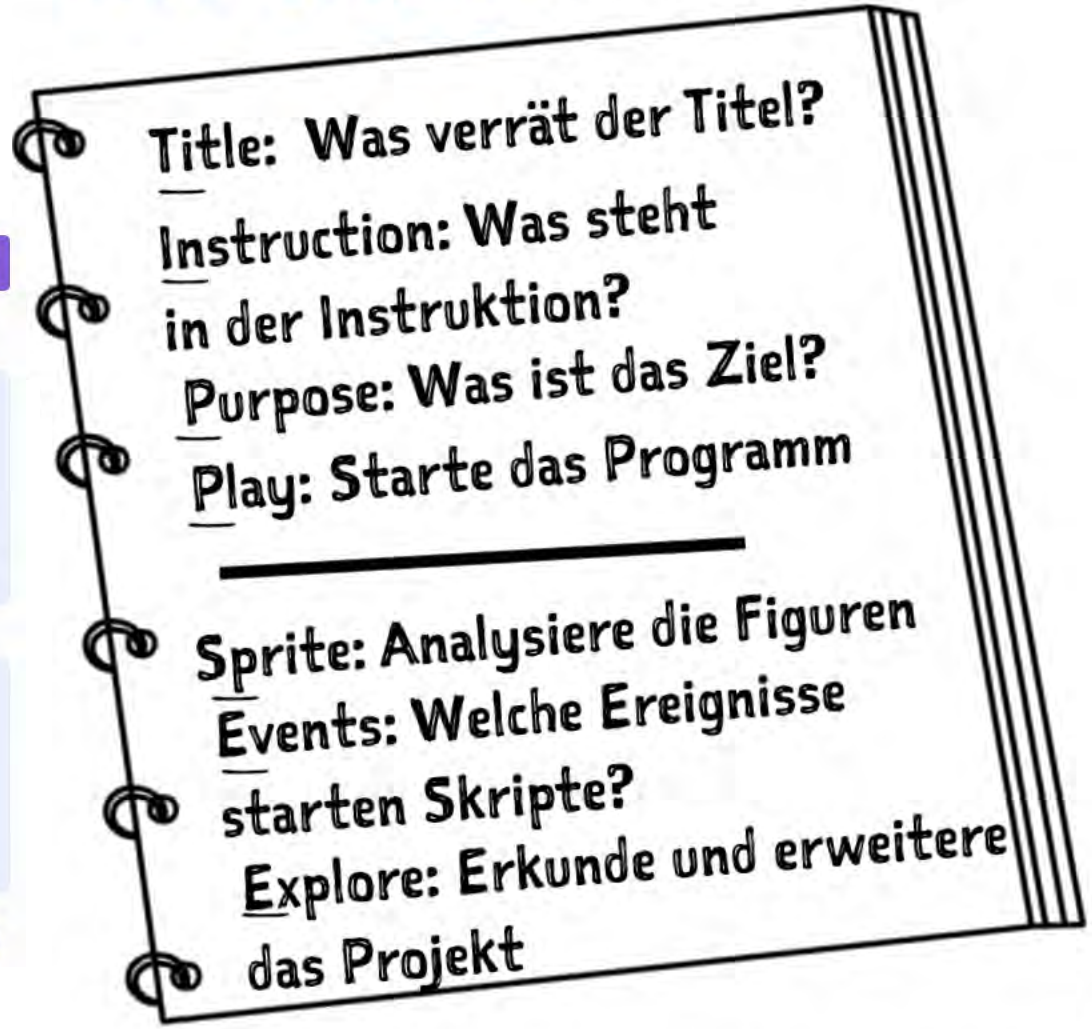
Predict-Beispiel

```
1.  jahr = int(input('Jahreszahl eingeben: '))
2.  if jahr % 400 == 0:
3.      print('Schaltjahr')
4.  elif jahr % 100 == 0:
5.      print('Kein Schaltjahr')
6.  elif jahr % 4 == 0:
7.      print('Schaltjahr')
8.  else:
9.      print('Kein Schaltjahr')
```

- „Stellt Vermutungen an, wozu das Programm dient. Überlege dir, mit welchen Eingabewerten du es prüfen willst.“

Eine PRIMM-Variante: TIPP & SEE für Scratch

https://scratch.mit.edu/projects/963295133/



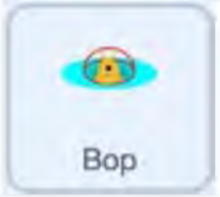
canonlab.org/scratch-encore

SEE ist inspiriert von Lesestrategien

Schau hinein auf Figuren und Ereignisse – Erkunde das Projekt:

Schau hinein:

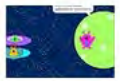
Notiere für jede **Figur** die Anzahl der Kostüme und kreise die **Ereignisse** ein, die Skripte haben.

Figur	Anzahl der Kostüme	Kreise die Ereignis-Blöcke ein, die über Skripte verfügen.
 Beep		
 Bop		

TIPP&SEE Modulübersicht

Modul 1: Scratch-Grundlagen

Modul 2: Ereignisse (Events)

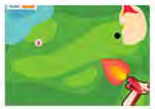


Modul 3: Animationen

Modul 4: Bedingte Wiederholung (Conditional Loops)

Modul 8: Variablen

Modul 12: Zerlegung nach Zweck

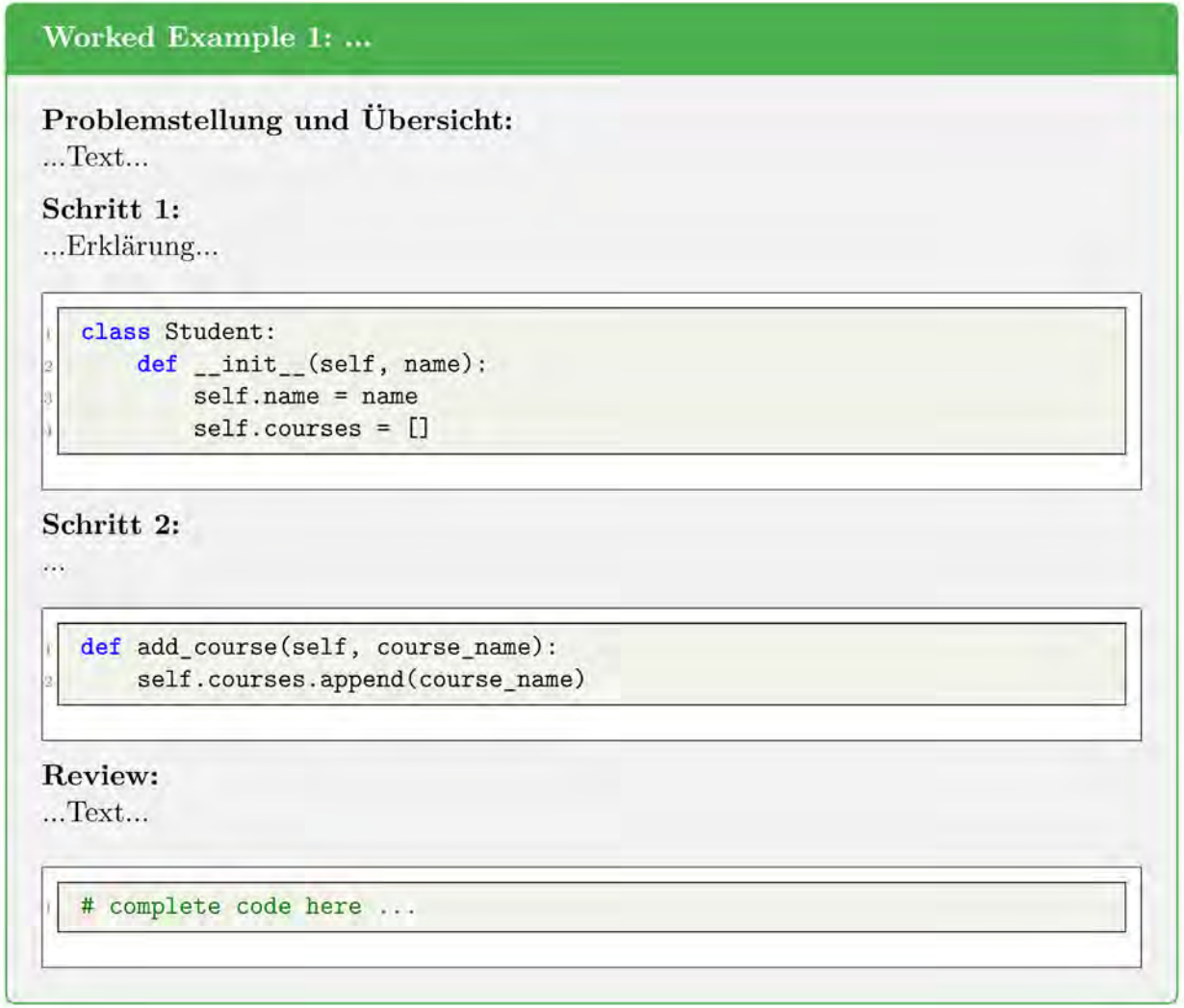


https://berufsinformatik.de/scratch/

https://www.canonlab.org/scratchencorematerials

Live Coding und Worked Examples

- Live Coding im Unterricht:
  - Lehrkräfte schreiben Code live und erläutern dabei ihr Vorgehen;
  - Fehler werden in Echtzeit korrigiert
- Ähnlich wirken Worked Examples
- Sie bestehen aus einer ***Aufgabenstellung*** und einer ***schrittweise kommentierten Lösung***, die den Problemlöseprozess transparent macht.



Worked Example 1: ...

Problemstellung und Übersicht:
...Text...

Schritt 1:
...Erklärung...

```
class Student:
    def __init__(self, name):
        self.name = name
        self.courses = []
```

Schritt 2:
...

```
def add_course(self, course_name):
    self.courses.append(course_name)
```

Review:
...Text...

```
# complete code here ...
```

Prompt

Erstelle ein *Worked Example* zur folgenden *Problemstellung* in [Programmiersprache]. Kennzeichne jeden Schritt eindeutig mit der Überschrift *Schritt X*. Jeder Schritt muss sowohl eine schriftliche Erklärung als auch Code enthalten und klar vom übrigen Inhalt abtrennbar sein. Es soll mindestens 3 und höchstens 10 Schritte geben. Füge außerdem am Anfang eine Übersicht (*Problemstellung und Übersicht*) ohne Code ein. In der Übersicht dürfen keine Schritte enthalten sein. Füge am Ende eine Zusammenfassung mit dem vollständigen Code ein. Die Ausgabe soll für Programmieranfänger*innen geeignet sein und Wert auf die Nutzung einfacher Konstrukte statt komplizierterer Konzepte legen. Verzichte auf Platzhalter (wie pass in Python). Dabei ist es in Ordnung, in einem Schritt nur eine einzelne Code-Zeile hinzuzufügen und zu erläutern. Konzeptuell zusammengehörige Zeilen sollten nicht aufgeteilt werden.

Beispielstruktur, an der du dich orientieren sollst: Problemstellung und Übersicht: Übersichtserklärung
Schritt 1: Erklärung zu Schritt 1
Code zu Schritt 1
Schritt 2: Erklärung zu Schritt 2
Code zu Schritt 2
Schritt 3: Erklärung zu Schritt 3
Code zu Schritt 3
Zusammenfassung: Erklärung
Vollständiger Code
Die Problemstellung lautet: [Problemstellung]

„Ein Python-Programm soll den größten Wert in einem 2D-Array finden.“

Problemstellung und Übersicht:
Die Aufgabe lautet: Ein **Python-Programm** soll den **größten Wert** in einem **zweidimensionalen Array (2D-Array)** finden und ausgeben.

Vorgehen:

- Wir legen ein **2D-Array** als Liste von Listen an.
- Wir setzen den **Startwert** für den größten Wert auf das erste Element.
- Wir durchlaufen mit **Schleifen** alle Elemente und aktualisieren den **größten Wert** bei Bedarf.
- Am Ende geben wir den **größten Wert** mit `print` aus.
  Wir verwenden bewusst einfache Konstrukte: Listen, Variablen, `for`-Schleifen, `if`-Vergleiche und `print`.

Schritt 1: Erklärung
Wir erstellen ein Beispiel-**2D-Array** (Liste von Listen). Jede innere Liste ist eine Zeile mit Zahlen.

Code zu Schritt 1

```python
array_2d = [
    [3, 5, 2],
    [9, 1, 4],
    [7, 6, 8]
]
```

Schritt 2: Erklärung
Wir wählen einen **Startwert** für den **größten Wert**. Dafür nehmen wir das erste Element im **2D-Array**. So vermeiden wir Spezialfälle und haben einen sinnvollen Vergleichswert.

Code zu Schritt 2

```python
groesster_wert = array_2d[0][0]
```

Schritt 3: Erklärung
Wir gehen mit zwei **for-Schleifen** durch alle Werte: zuerst durch jede Zeile, dann durch jedes Element in der Zeile. Mit einem **if-Vergleich** prüfen wir, ob ein gefundener Wert größer ist als der aktuelle **größte Wert**. Falls ja, aktualisieren wir die Variable.

Code zu Schritt 3

```python
for zeile in array_2d:
    for wert in zeile:
        if wert > groesster_wert:
            groesster_wert = wert
```

Schritt 4: Erklärung
Zum Schluss geben wir den **größten Wert** aus. So sehen wir das Ergebnis auf der Konsole.

Code zu Schritt 4

```python
print("Der größte Wert ist:", groesster_wert)
```

Zusammenfassung: Erklärung
Wir haben ein **2D-Array** definiert, einen **Startwert** gesetzt, alle Elemente mit **Schleifen** untersucht, bei größerem Wert den **Vergleich** gewonnen und am Ende das Ergebnis mit `print` ausgegeben. Das Programm nutzt nur grundlegende Python-Konzepte und ist dadurch gut nachvollziehbar.

Vollständiger Code

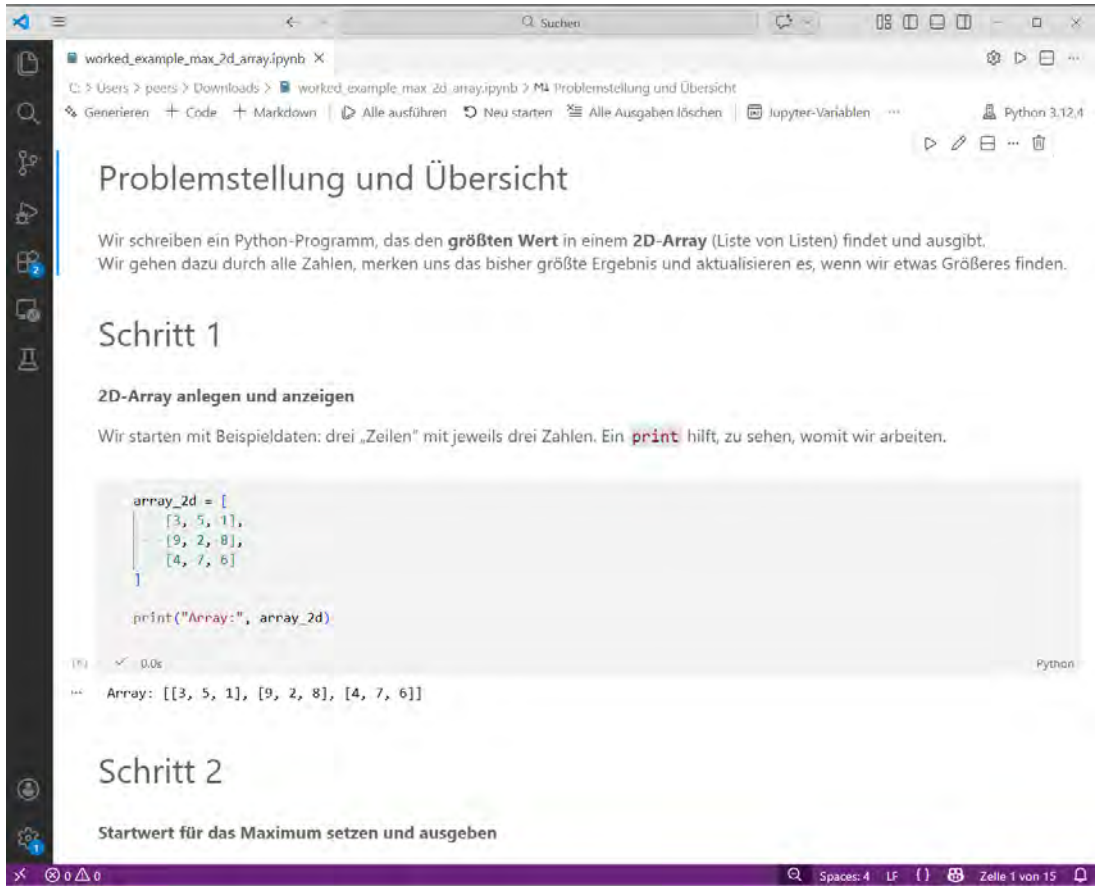
```python
array_2d = [
    [3, 5, 2],
    [9, 1, 4],
    [7, 6, 8]
]

groesster_wert = array_2d[0][0]
```

Computational Notebooks

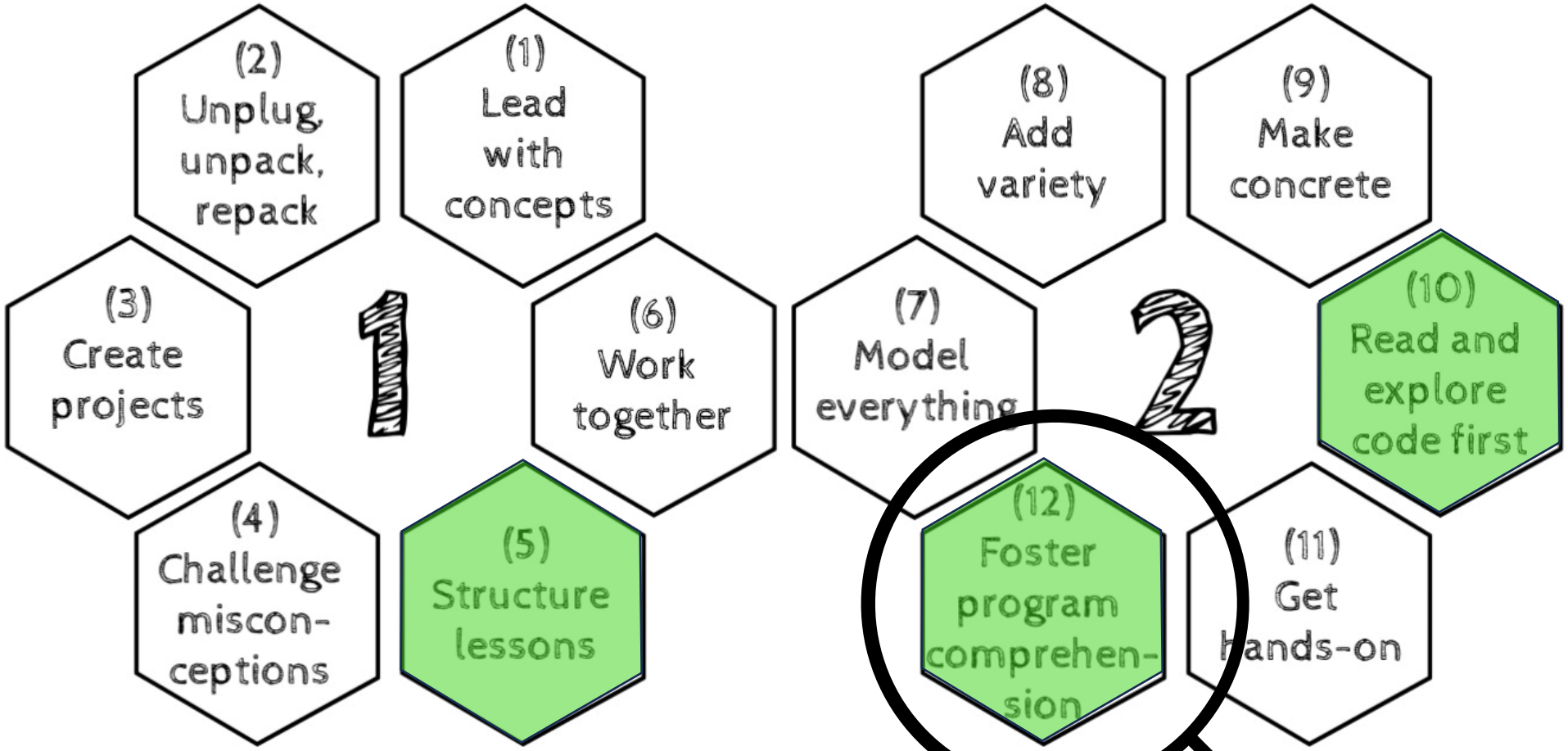
(z.B. Jupyter Notebooks)

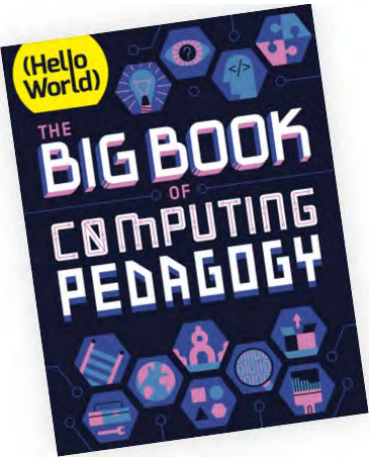
- Kombination von
  - ausführbarem Programmcode,
  - beschreibendem Text und
  - Programmausgabe in einem Dokument
- Der Split-Attention-Effect empfiehlt, zusammenhängende Informationen in einem integrierten Format zu präsentieren



Jupyter Notebook in Visual Studio Code zum Worked Example

12 Fachdidaktische Prinzipien des NCCE





https://youtu.be/IWRGHdgntog

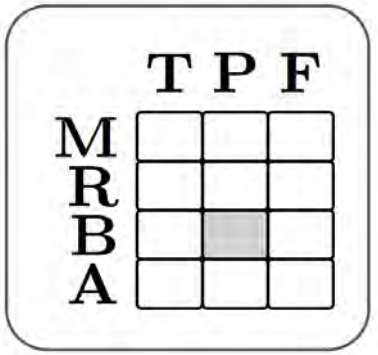
Prinzip 3: Programmverstehen systematisieren (Blockmodell)

- „Artefakte“ können auf ihre Struktur (T) und ihre Funktion (F) untersucht werden
- Programme zusätzlich auf ihr Verhalten während der Ausführung (P)

	Textoberfläche (T)	**Programm-Ausführung (P)**	**Funktion/Intention (F)**
Makro-Ebene (M)	Das Programm als Ganzes auf der Textoberfläche erfassen.	Den algorithmischen Ablauf verstehen.	Den Zweck / das Ziel des Gesamtprogramms im Anwendungs-kontext formulieren.
Relation (R)	Beziehungen zwischen Blöcken identifizieren.	Zusammenspiel von syntaktischen und semantischen Einheiten bei der Ausführung erklären.	Zusammenhänge zwischen Teilzielen des Programms erkennen.
Block (B)	Syntaktische oder semantische Einheiten (Blöcke) im Code erkennen.	Blöcke als Sequenz von Anweisungen verstehen.	Den Zweck eines Code-Blocks erklären.
Atom (A)	Einzelne Sprachelemente in einer Zeile identifizieren.	Ausführen eines Elements.	Den Zweck einer einzelnen Zeile erläutern.

in Anlehnung an Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., & Paterson, J. H. (2010). An introduction to program comprenension for computer science educators.

Einordnung in das Blockmodell: Tracing



Ein einfacher „Python-Computer"

(in Anlehnung an Mühling, 2020)

```
1.  jahr = int(input('Jahreszahl eingeben: '))
2.  if jahr % 400 == 0:
3.      print('Schaltjahr')
4.  elif jahr % 100 == 0:
5.      print('Kein Schaltjahr')
6.  elif jahr % 4 == 0:
7.      print('Schaltjahr')
8.  else:
9.      print('Kein Schaltjahr')
```

Aktuelle Zeile:

7

globaler Variablen-Speicher:

Variablen-Name:	Wert:
jahr	2024

Objekt-Speicher:

Objekt-Name:	Wert:

lokaler Speicher (pro Funktion)

Variablen-Name:	Wert:

return:

Bedingung:

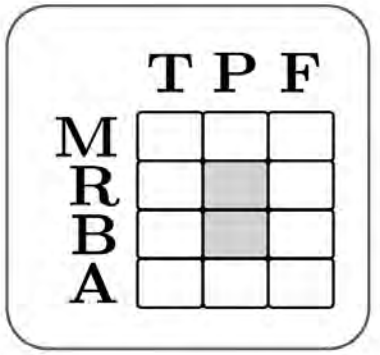
```
jahr % 4 == 0
true
```

Konsole:

```
>>>
Jahreszahl eingeben:
2024
Schaltjahr
```

(vgl. Xie, Nelson & Ko, 2017)
(vgl. Stechert, 2020b)

Einordnung in das Blockmodell: Tracing mit Funktionsaufrufen



Ein einfacher „Python-Computer" mit Funktionsaufrufen

Anwendungsbeispiel:
Eintragen eines Termi s von 8 bis 14 Uhr

```
1  # Wiederholte Eingabe, bis Ganzzahl zwischen von und bis
2  # eingegeben wird, und gibt die eingegebene Zahl zurück
3  def zahl_eingeben(von, bis):
4      fertig = False
5      while not fertig:
6          zahl = int(input())
7          if zahl >= von and zahl <= bis:
8              fertig = True
9          else:
10             print('Bitte eine Zahl zwischen', von,  'und',
                   bis, 'eingeben!')
11     return zahl

12 # Hauptprogramm: Dauer eines Termins berechnen
13 print('Wann beginnt der Termin vormittags?')
14 von = zahl_eingeben(6, 12)
15 print('Wann endet der Termin nachmittags?')
16 bis = zahl_eingeben(13, 18)
17 print('Dauer des Termins in Stunden:', bis - von)
```

Globaler Speicher (Hauptprogramm):

Variablen-Name:	Wert:
von	8
bis	14

Lokaler Speicher (Funktion zahl_eingeben()):

Variablen-Name:	Wert:
von	13
bis	18
fertig	True
zahl	14
return:	14

Aktuelle Zeile:

17

Bedingung:

not fertig

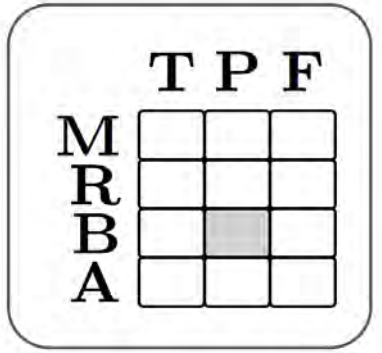
False

Konsole:

```
>>>
Wann beginnt der Termin vormittags?
8
Wann endet der Termin nachmittags?
14
Dauer des Termins in Stunden: 6
```

(in Anlehnung an Mühling, 2020, sowie Xie, Nelson & Ko, 2017) (vgl. Stechert, 2022)

Einordnung in das Blockmodell: Parsons Problems



Temperatur-Puzzle

```
else :
if temperature < 20 :
elif temperature > 25 :
print("Es ist zu kalt.")
temperature = float(input("Bitte geben Sie die aktuelle Temperatur ein: "))
print("Es ist zu heiß")
print("Es ist angenehm warm.")
```

Get Feedback | Reset Problem

parsons-puzzles is maintained by PeerStechert. This page was generated by GitHub Pages.

Haynes & Ericson, 2021

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems.

https://peerstechert.github.io/parsons-puzzles/

Das Blockmodell füllen: Das „I“ in PRIMM

Was passiert, wenn zwei Zeilen vertauscht werden?

Was passiert, wenn die Eingabe ____ ist?

Code kommentieren

Kontrollfluss zeichnen (z. B. Sprung aus einer Schleife)

Einzelnes Konstrukt identifizieren, z.B. Funktionsaufruf

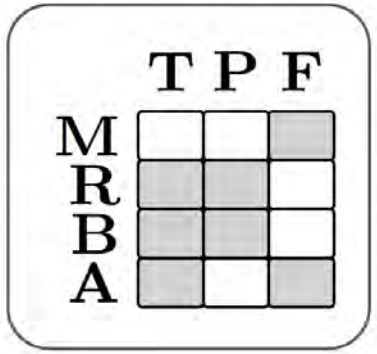
Gültigkeits-bereich einer Variablen bestimmen

Den Zweck einer Anweisung bestimmen

Sentance: The I in PRIMM. Hello World: The Big Book of Computing Pedagogy. 2020

Das „I“ in PRIMM

- Programmcode erklären und tracen



	Textoberfläche (T)	Programm-Ausführung (P)	Funktion/Intention (F)
Makro-Ebene (M)			*Frage, „Was passiert, wenn die Eingabe _____ ist?"*
Relation (R)	*Den Gültigkeitsbereich (Scope) einer Variablen bestimmen.*	*Den Kontrollfluss zeichnen.*	
Block (B)	*Code annotieren: Blöcke oder unterschiedliche Konstrukte identifizieren.*	*Frage: „Was passiert, wenn diese beiden Zeilen vertauscht werden?“;* *Den Kontrollfluss zeichnen.*	
Atom (A)	*Ein einzelnes Konstrukt im Programm identifizieren, z. B. „Finde und markiere einen Funktionsaufruf.“):*		*Den Zweck einer einzelnen Anweisung bestimmen.*

Tabelle 5.3: Fragestellungen der „Investigate-Phase” im PRIMM-Modell in Anlehnung an Sue Sentance, 2020

Fazit

- **Programmieren bleibt wichtig – aber verändert sich**

→ Fokus verschiebt sich von Schreiben → Verstehen & Bewerten

- **Programmieren ist schwer – wegen kognitiver Belastung**

→ Unterricht muss aktiv entlasten

- **Guter Unterricht ist strukturiert**

→ z. B. PRIMM / Use-Modify-Create + gezielte Aufgabenformate

Ausblick 1: Themen & Kontexte

Beispiel:

- Der neue Orientierungsrahmen Globale Entwicklung – Bildung für nachhaltige Entwicklung in der gymnasialen Oberstufe

https://www.engagement-global.de/de/orientierungsrahmen

Ausblick 2: Königstein-AG „Neue Prüfungsformate“

(18.03.2026-20.03.2026)

Auslöser:

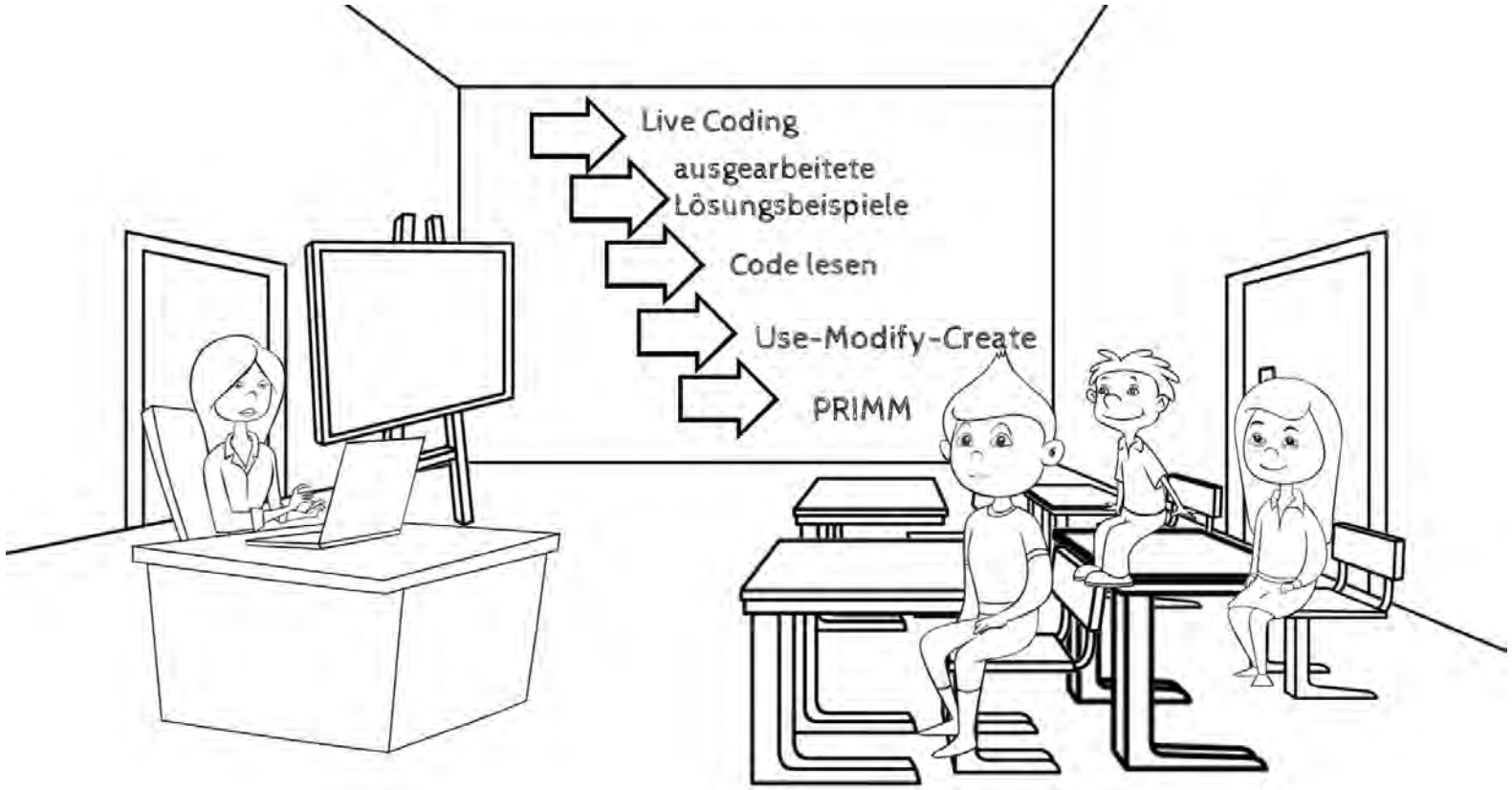
- „Es wird medien- und materialgestütztes Arbeiten unter Aufsicht als neue Form eines Prüfungsformates ermöglicht.
- Medien und Materialien können beispielsweise
  - analoge und digitale Nachschlagewerke,
  - eigene Aufzeichnungen der Schülerinnen und Schüler,
  - digitale Geräte mit Internetzugang,
  - Programme zur Textverarbeitung oder Tabellenkalkulation,
  - Zeichensoftware oder
  - KI-basierte Anwendungen wie Large Language Models sein.“

Foto: Wolf Spalteholz

FACHAUSSCHUSS IBS

Leistungsnachweise in der Sekundarstufe I. Schleswig-Holsteinischer Erlass vom 4. Juni 2025: https://t1p.de/jtsp2

Vielen Dank für die Aufmerksamkeit



YouTube-Kanal:
https://t1p.de/tygpc

Prompt & Folien:
https://t1p.de/d5x2q