FAIBLE.nrw

# Didaktik der Programmierung

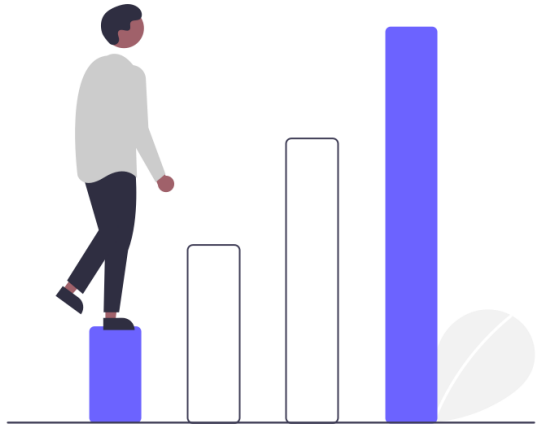
UNIVERSITÄT PADERBORN


Notional Machine
Modellieren
Projektunterricht
Bildung
Pair Programming
PRIMM
Programmiersprachen
Software Life Cycle
4C/ID
Kompetenzen
Tools
Softwareentwicklung
Problemfelder
Lernroboter
Implementieren
Unterrichtsmethoden
Lernerfolg- und Leistungsbewertung
Worked Example
Inhalte
Lernziele
Programmverstehen
Digitale Welt
Debugging
Cognitive Load
STREAM
Block-basierte Sprachen
Computational Thinking
Programmieren
Use-Modify-Create
Programmierkonzepte
Computational Notebooks
Softwareprojekte
Lehrpläne

# Kapitelübersicht

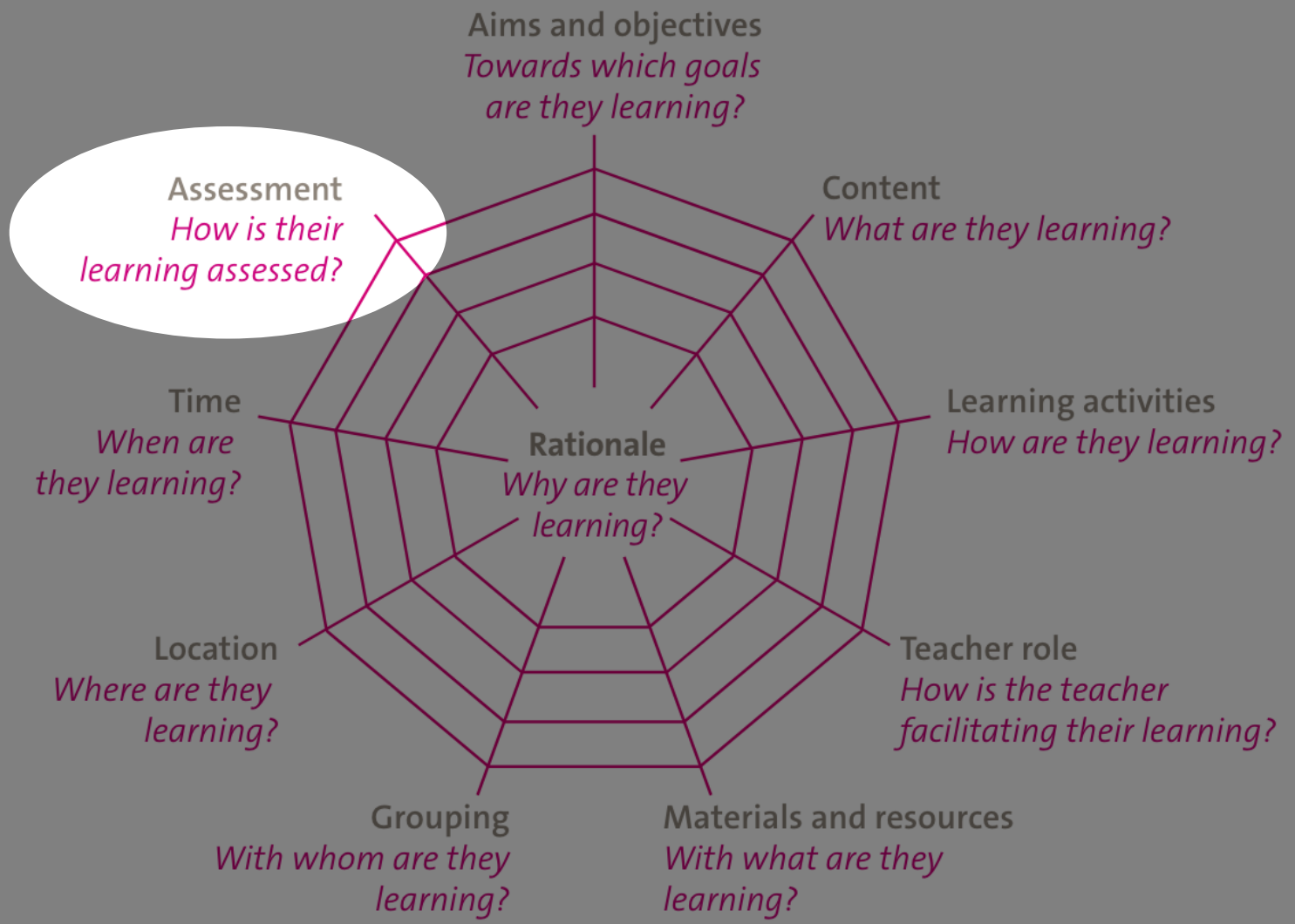
1. Einführung
2. Ziele und Inhalte
3. Lerntheorien
4. Unterrichtsplanung
5. Programmiersprachen und Umgebungen
6. Programmieren im Team
7. **Lernerfolgskontrolle und Leistungsbewertung**

# 7. Lernerfolgskontrolle und Leistungsbewertung



Wie lässt sich der Lernerfolg messen?

Nach welchen Kriterien sollen die Leistungen der Schüler:innen bewertet werden?


Aims and objectives
Towards which goals are they learning?
Content
What are they learning?
Learning activities
How are they learning?
Teacher role
How is the teacher facilitating their learning?
Materials and resources
With what are they learning?
Grouping
With whom are they learning?
Location
Where are they learning?
Time
When are they learning?
Assessment
How is their learning assessed?
Rationale
Why are they learning?

# Selbstkontrolle & Automatisierte Tests

- Automatisierte Tests (Unit Tests)
  - Ermöglicht eigenständige Überprüfung der Lösung
  - Entlastung der Lehrkraft
  - Transparente und objektive Bewertungskriterien
    - Tests stellen formalisierte Anforderungen dar
- KI-Assistent
  - Kann auf Fehler hinweisen oder gezielte Hilfestellung geben, ohne die Lösung vorwegzunehmen

# Beispiel

**Aufgabenstellung**

Schreiben Sie ein Python-Programm, das eine Funktion ist_primzahl(n) enthält. Diese Funktion soll überprüfen, ob eine gegebene Zahl n eine Primzahl ist oder nicht. Eine Primzahl ist eine natürliche Zahl größer als 1, die nur durch 1 und sich selbst teilbar ist.

**Anforderungen:**

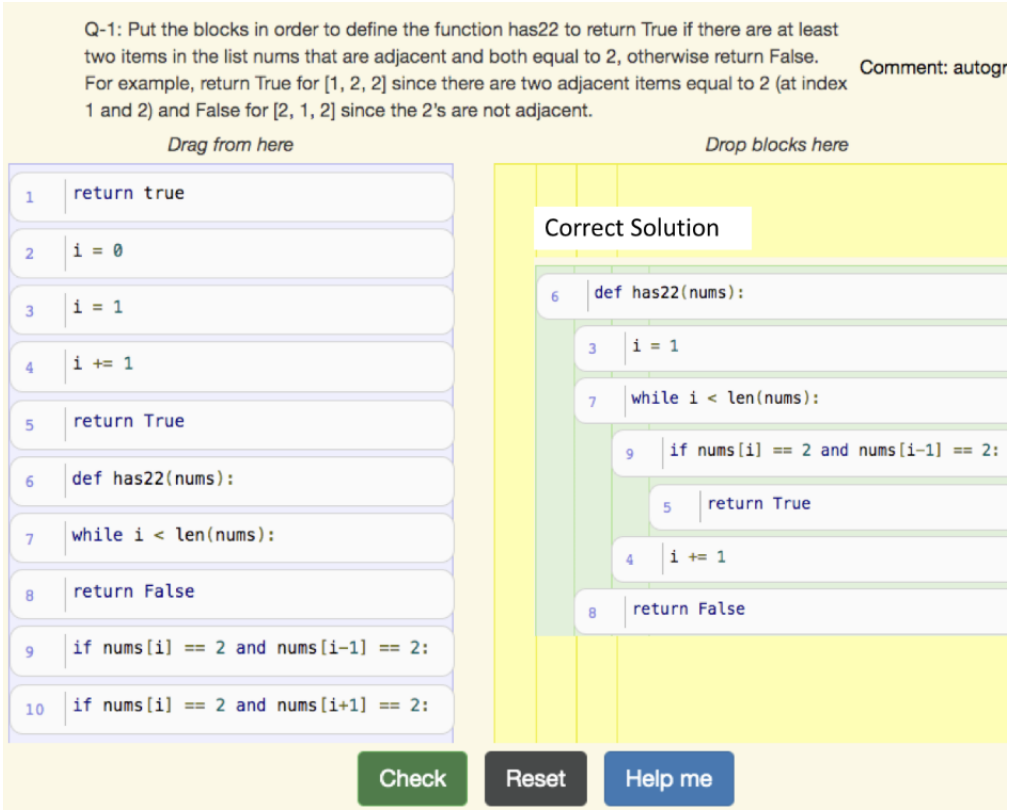
1. Die Funktion ist_primzahl(n) soll True zurückgeben, wenn n eine Primzahl ist, andernfalls False.
2. Die Funktion soll für alle Ganzzahlen n funktionieren, inklusive negativer Zahlen und 0.

```python
def ist_primzahl(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

assert(ist_primzahl(3))
assert(ist_primzahl(5))
assert(ist_primzahl(7))
assert(not ist_primzahl(8))
assert(not ist_primzahl(9))
assert(not ist_primzahl(10))
```

# Parsons Problems

- Fragmentierter Programmcode
- Ziel: Codefragmente in die richtige Reihenfolge bringen
- Optional: Nicht benötige oder inkorrekte Codefragmente zur Ablenkung (Distractors)
- Guter Prädiktor für die Programmierfähigkeit *(Denny et al., 2008)*
- Ermöglicht schnelle und objektive Bewertung




*Haynes & Ericson, 2021*

Haynes, C. C., & Ericson, B. J. (2021). Problem-Solving Efficiency and Cognitive Load for Adaptive Parsons Problems vs. Writing the Equivalent Code.
Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems.

# Expertise Reversal Effect

*(vgl. Kapitel „Lerntheorien“)*

- Parsons-Probleme sind möglicherweise ungeeignet, um die Programmierfähigkeiten erfahrener Programmierer:innen zu überprüfen
- (Micro) Parsons Problems werden von Experten teilweise als schwieriger wahrgenommen als „offene“ Programmieraufgaben. *(Wu & Smith, 2024)*

Wu, Z., & Smith, D. H. (2024). Evaluating Micro Parsons Problems as Exam Questions.

# Checkliste: OO Programmierkonzepte

*(Sanders & Thomas, 2007)*

| **Some things to look for** | **Suggests understanding of:** |
|---|---|
| ☑ Program compiles | Basic mechanics |
| ☑ Constructors defined and used | Constructors |
| ☑ Multiple instances of same class | Object and class |
| ☑ Variables / methods with same name in different classes | Encapsulation |
| ☑ Multiple classes defined in program<br>☑ Composite object (object with parts) defined<br>☑ Object passed as parameter to constructor (peer object)<br>☑ Peer object assigned to instance variable<br>☑ Methods other than constructors defined<br>☑ Message sent to part / peer object<br>☑ Methods' return values used | Linking; message passing methods |
| ☑ Library classes extended<br>☑ User-defined classes extended<br>☑ Shared properties/methods factored into superclass | Inheritance |
| ☑ Single library class has multiple subclasses that define the same method differently<br>☑ Single user-defined class has multiple subclasses that define the same method differently | Polymorphism |
| ☑ Classes correspond to visible objects in domain<br>☑ Some class corresponds to invisible (conceptual) object | Modelling |
| ☑ Methods have parameters<br>☑ Method parameters used<br>☑ Inheritance hierarchies defined and used<br>☑ Simple recipes used<br>☑ Design patterns used | Abstraction |

Sanders, K., & Thomas, L. (2007). Checklists for grading object-oriented CS1 programs: Concepts and misconceptions.

# Indikatoren für Fehlvorstellungen

*(Sanders & Thomas, 2007)*

| **Some things to look for** | **Suggests misconception:** |
|---|---|
| ☑ Classes identical except for class names<br>☑ Classes identical except for property values that could be set in constructors<br>☑ Classes identical except for very minor changes<br>☑ Classes rarely/never instantiated more than once<br>☑ Superclass/subclass used instead of class/instance | Instance / class conflation (Holland) |
| ☑ All code in a single class<br>☑ Code in single class instead of composite class and parts<br>☑ Classes defined but not linked in<br>☑ Work in methods exclusively done by assignment<br>☑ Objects passed as parameters but not used | Problems with linking and interaction (Thomasson et al.) |
| ☑ Class whose instances all model elements of some collection | Class / collection conflation (Thomasson et al.) |
| ☑ Classes identical except for property values that could be set in constructors<br>☑ Duplicate code not factored into superclass<br>☑ Duplicate method signatures in different classes not defined as interface | Problems with abstraction (Or-Bach and Lavy) |
| ☑ Classes joined that should be separate or vice versa<br>☑ Classes do not correspond to objects in domain<br>☑ Failure to use inheritance to model hierarchical domain | Problems with modelling (Thomasson et al.; Eckerdal & Thuné) |
| ☑ Failure to delete irrelevant code when adapting | Classes just text (Eckerdal & Thuné) |
| ☑ Variables with names that are really values of attributes | Identity/attribute conflation (Holland et al.) |
| ☑ No classes with methods other than constructors or accessor/mutators (or `main`) | Objects are only data records (Holland et al.) |
| ☑ Methods/variables in different classes always have different names | Problems with encapsulation (Fleury) |

Sanders, K., & Thomas, L. (2007). Checklists for grading object-oriented CS1 programs: Concepts and misconceptions.

# Illustrationen

- Illustrationen von Limpitsouni, K. unter freier Lizenz via https://undraw.co

FAIBLE.nrw

Beteiligte Hochschulen:

- RWTH-Aachen
- Westfälische Wilhelms-Universität Münster
- Universität Duisburg-Essen
- Universität Bonn
- Universität Paderborn
- Technische Universität Dresden
- Carl von Ossietzky Universität Oldenburg

ORCA.nrw
Das Landesportal für Studium und Lehre.

Ein Kooperationsvorhaben empfohlen durch die:
DIGITALE HOCHSCHULE NRW
INNOVATION DURCH KOOPERATION

gefördert durch:
Ministerium für Kultur und Wissenschaft des Landes Nordrhein-Westfalen