Didaktik der Programmierung


Notional Machine
Modellieren
Projektunterricht
Pair Programming
PRIMM
Bildung
Programmiersprachen
Software Life Cycle
4C/ID
Kompetenzen
Tools
Softwareentwicklung
Problemfelder
Lernroboter
Implementieren
Unterrichtsmethoden
Lernerfolg- und Leistungsbewertung
Worked Example
Lernziele
Inhalte
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

5 Problemfelder (Areas of Difficulty)

(Du Boulay, 1986)

1. Orientierung
   - “finding out what programming is for, what kinds of problem can be tackled and what the eventual advantages might be of expending effort in learning the skill”
2. Programmausführung (Notional Machine)
   - “how the behavior of the physical machine relates to this notional machine”
3. Notation
   - “various formal languages”
   - “mastering the syntax and the underlying semantics”
4. Standard-Strukturen, Klischees oder Pläne
   - “that can be used to achieve small-scale goals, such as computing a sum using a loop“
5. Pragmatik
   - “how to specify, develop, test, and debug a program using whatever tools are available“

Du Boulay, B. (1986). Some difficulties of learning to program.

Kognitive Belastung beim Programmieren lernen

- Programmieren lernen fällt vielen Schüler:innen schwer
  - Hoher wahrgenommener inhärenter Schwierigkeitsgrad *(Baldwin & Kuljis, 2000)*
  - Hohe Abbruchquote in Programmierkursen an Hochschulen *(Baldwin & Kuljis, 2000)*
  - Viele neue, abstrakte und voneinander abhängige Konzepte
  - Einsatz von Tools mit zahlreichen Funktionalitäten (Debugger, Versionskontrolle, Plugins)

→ Hohe kognitive Belastung!

Baldwin, L. P. & Kuljis, J. (2000). Visualisation techniques for learning and teaching programming.

Lösungsansätze (Beispiele)

- Verschiedene Ansätze für den Einstieg in die Programmierung
  - Spiralansatz
  - Expertenansatz
  - **Leseansatz**
- Programmablauf
  - **Notional Machines** als didaktisches Werkzeug
  - **Python Tutor**
- Worked Examples
  - Adaption von Beispielen
- Block-basierte Sprachen
  - Vermeidung von Syntax-Fehler

(New) Cognitive Load Theory

Kognitive Belastung beim Programmieren

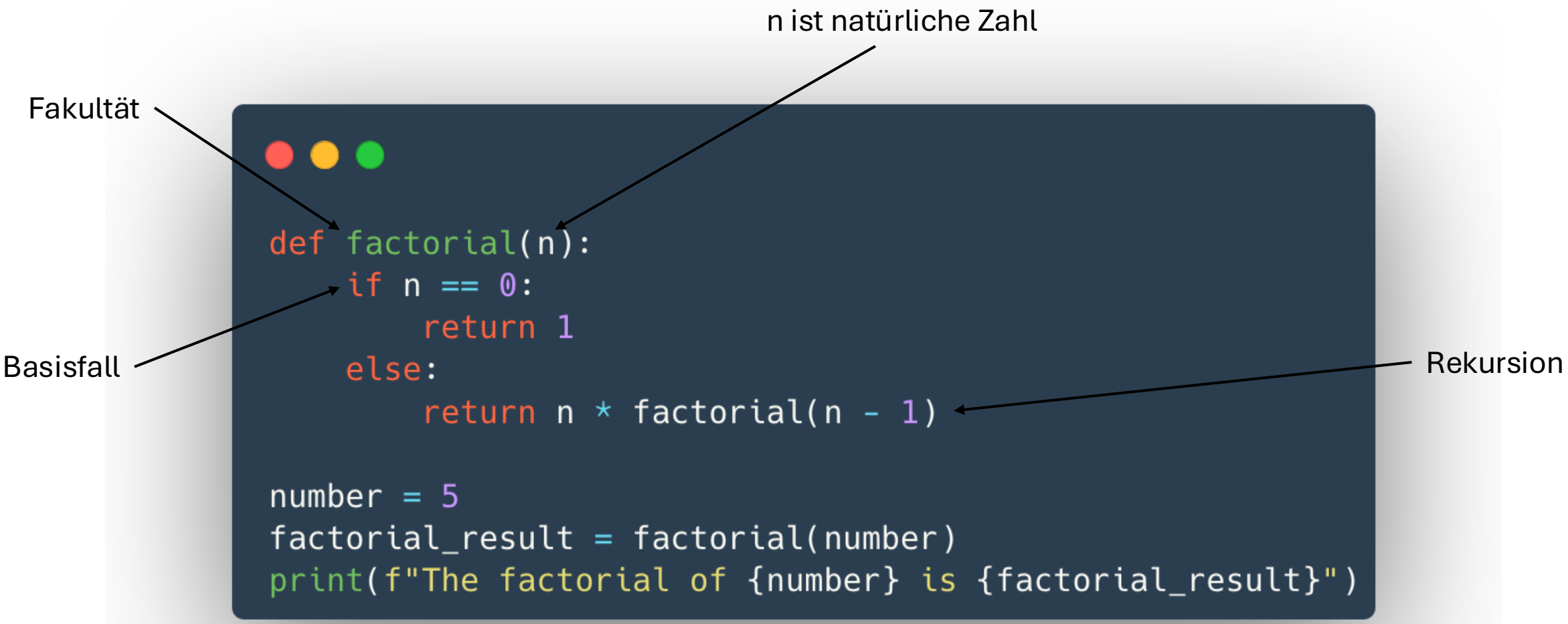
Cognitive Architecture

(Sweller et al., 1998)

- Langzeitgedächtnis (Long-term Memory)
  - Praktisch unbegrenzte Kapazität
- Arbeitsgedächtnis (Working Memory)
  - Kapazität und Speicherdauer stark begrenzt
  - Für die Informationsverarbeitung zuständig
  - „Flaschenhals“
- Schemata
  - Domänenspezifisches Wissen wird im Langzeitgedächtnis in sogenannten Schemata gespeichert
  - Einmal gelernt, kann selbst ein komplexes Schema abgerufen und als einzelnes Element im Arbeitsgedächtnis behandelt werden

Sweller, J.; Van Merrienboer, J. J. & Paas, F. G. (1998). Cognitive architecture and instructional design.

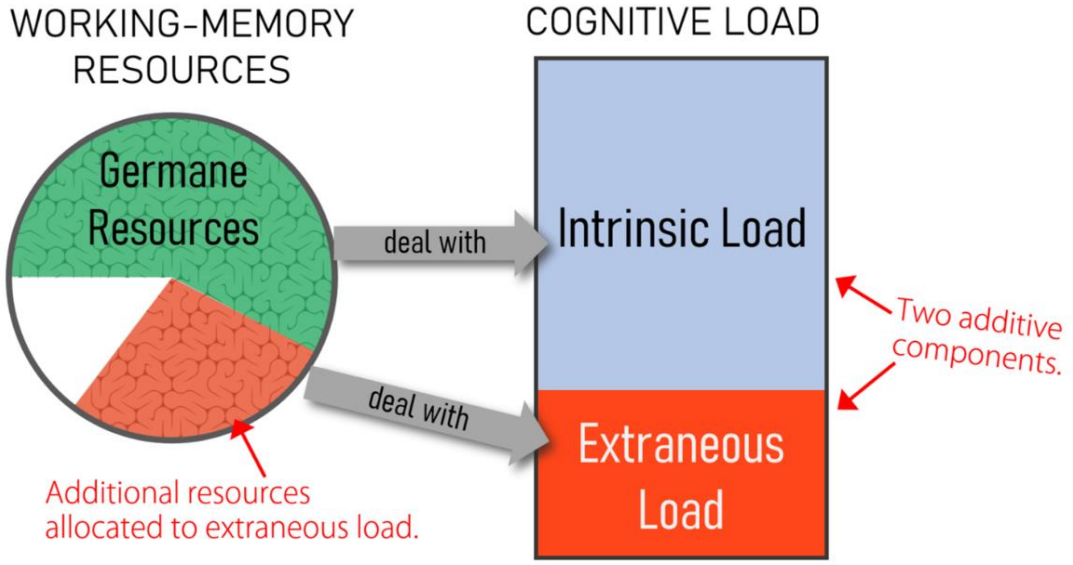
Beispiele für Schemata



Kognitive Belastung I

(Duran et al., 2022)

- Intrinsische Belastung
  - Kognitiven Belastung, die durch das Lernmaterial selbst bedingt wird (Schwierigkeitsgrad der Aufgabe)
- Nicht-lernbezogene *(extraneous)* Belastung
  - Kognitive Belastung, die durch die Darstellung des Lerninhalts bedingt wird (z.B. überflüssige Informationen)
- Lernbezogene (*germane)* Ressourcen
  - Ressourcen des Arbeitsspeichers, zur Bewältigung der intrinsischen kognitiven Belastung verwendet werden (unter Verwendung von Schemata)



(Duran et al., 2022)

Duran, R., Zavgorodniaia, A., & Sorva, J. (2022). Cognitive Load Theory in Computing Education Research: A Review.

Kognitive Belastung II

(Duran et al., 2022)

- Lernen findet in der Auseinandersetzung mit der intrinsischen Belastung statt
- Übersteigt die kognitive Belastung die kognitiven Ressourcen, kommt es zu einer Überlastung und zu einer Abnahme der Lerneffektivität (Cognitive Overload).
- Schemata reduzieren die benötigten lernbezogenen Ressourcen
- Ziel: Verringerung der nicht lernbezogenen Belastung
- Annahme der Theorie: Motivation des Lerners vorhanden

Duran, R., Zavgorodniaia, A., & Sorva, J. (2022). Cognitive Load Theory in Computing Education Research: A Review.

Effekte und Konsequenzen für den IU

(Sweller et al., 1998; Duran et al., 2022)

- Worked-example effect
  - Die Betrachtung vollständig gelöster Beispiele kann die Lernleistung verbessern, insbesondere bei der Lösung ähnlicher Probleme
- Split-attention effect
  - Zusammenhängende Informationen in einem integrierten Format präsentieren, damit das Arbeitsgedächtnis des Lernenden nicht durch die Verknüpfung der Informationen belastet wird (z.B. Computational Notebooks)
- Redundancy effect
  - Ein und dieselbe Information nicht in mehreren in sich geschlossenen Formaten präsentieren
- Isolated-elements effect
  - Teile sehr komplexer Informationen zunächst isoliert behandeln, bevor sie integriert werden
- Self-explanation effect
  - SuS Konzepte zunächst sich für sich selbst erklären lassen
- Modality Effect
  - Informationen, die verbal präsentiert werden (gesprochen), können besser behalten werden als Informationen, die visuell präsentiert werden

Sweller, J.; Van Merrienboer, J. J. & Paas, F. G. (1998). Cognitive architecture and instructional design.

Zusammengesetzte Effekte

(Sweller et al., 1998; Duran et al., 2022)

- Element interactivity effect
  - Die zuvor genannten Effekte gelten nur, wenn der präsentierte Inhalt ausreichend komplex ist (individuell abhängig)
- Expertise reversal effect
  - Effekte verschwinden oder kehren sich bei Expert:innen sogar um

Sweller, J.; Van Merrienboer, J. J. & Paas, F. G. (1998). Cognitive architecture and instructional design.

Übung

Überlegen Sie sich für **mindestens drei** der genannten **Effekte** ein **Beispiel** aus dem Unterricht (Programmierung), bei dem sich der jeweilige Effekt **zu Nutze gemacht** werden kann!

Einzelarbeit

10 Minuten

Ansätze zur Verringerung der kognitiven Belastung beim Programmieren lernen

- Strukturierung der Lerninhalte (Curricular)
  - Reading-first
  - Spiralansatz
- Methodik
  - 4C/ID Modell *(siehe Kapitel 4)*
  - Pair Programming *(siehe Kapitel 6)*
    - Der Navigator behält den Kopf frei
  - Worked Example
    - Orientierung an Beispielen
- Verwendung didaktischer Programmiersprachen oder –Umgebungen *(siehe Kapitel 5)*
  - z.B. Block-basierte Sprachen wie Scratch (Syntax fällt weg)

Leseansatz

zum Programmieren lernen

Programmcode lesen, erklären und schreiben

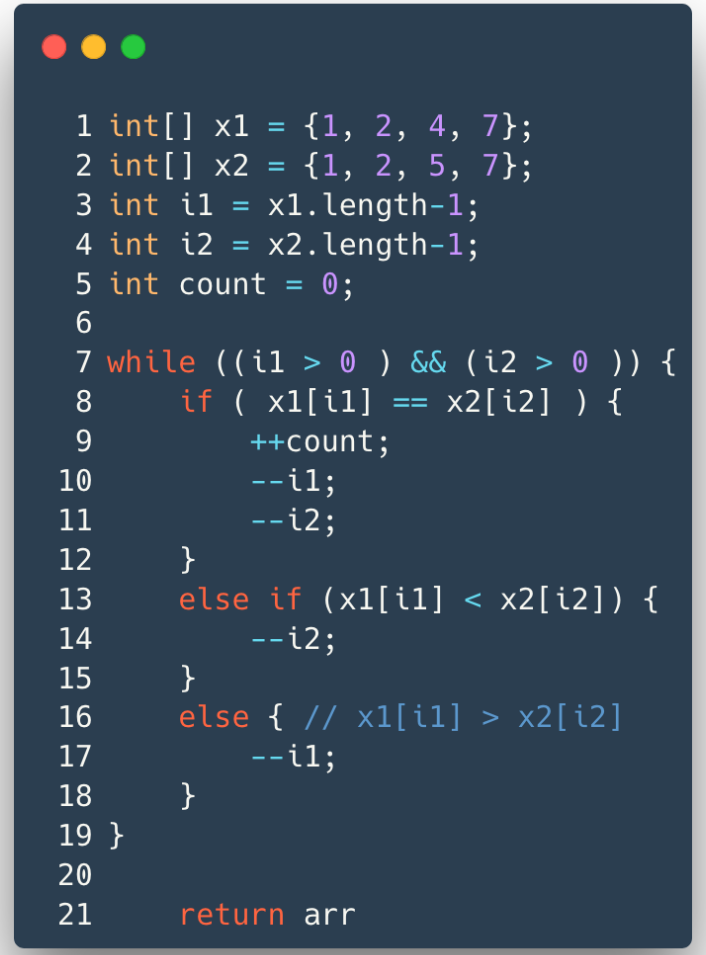
(Lister, 2004; Lopez et al., 2008)

- Untersuchungen über den Zusammenhang von **Lesen**, **Erklären** und **Schreiben** von Code *(Lister, 2004; Lopez et al., 2008)*
- Code tracing + Code explaining = Code writing *(Lister, 2022)*
  - Starke Korrelation zwischen der Fähigkeit Programmcode nachzuzeichnen (Tracing) und zu erklären und der Fähigkeit Programmcode zu schreiben

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., & Thomas, L. (2004). A Multi-National Study of Reading and Tracing Skills in Novice Programmers.
Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming.

Programmcode nachzeichnen (Tracing)

(Lister, 2022)



```
1 int[] x1 = {1, 2, 4, 7};
2 int[] x2 = {1, 2, 5, 7};
3 int i1 = x1.length-1;
4 int i2 = x2.length-1;
5 int count = 0;
6
7 while ((i1 > 0 ) && (i2 > 0 )) {
8     if ( x1[i1] == x2[i2] ) {
9         ++count;
10        --i1;
11        --i2;
12    }
13    else if (x1[i1] < x2[i2]) {
14        --i2;
15    }
16    else { // x1[i1] > x2[i2]
17        --i1;
18    }
19 }
20
21    return arr
```

Welchen Wert hat die Variable count nach Ausführung des Programms?

a) 3

b) 2

c) 1

d) 0

Lister, R. (2022). Some thoughts on designing eye movement studies for novice programmers.

Programmcode erklären

(Lister, 2022)

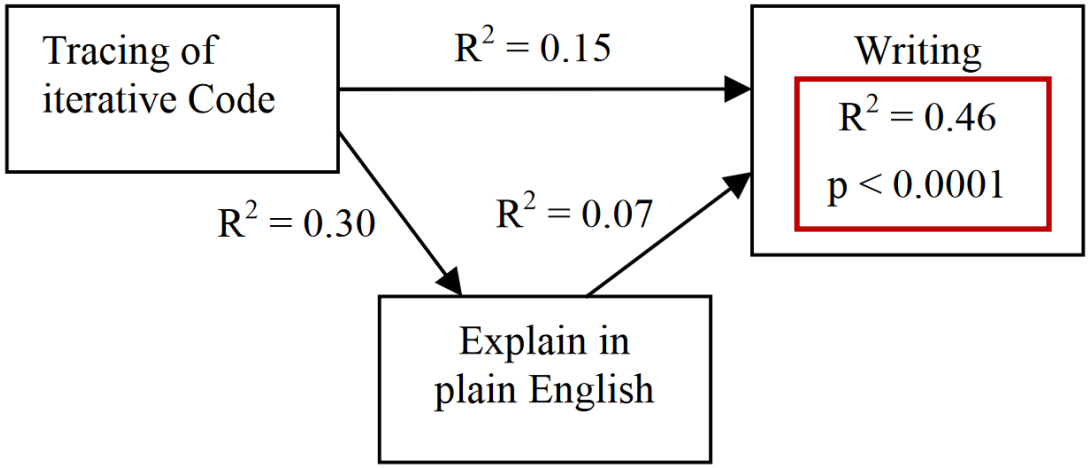
```
1 def enigma(nums: list):
2     for index in range(len(nums) - 1):
3         if nums[index] > nums[index + 1]:
4             return False
5     return True
```

Erkläre in ein bis zwei Sätzen, was die Funktion enigma macht!

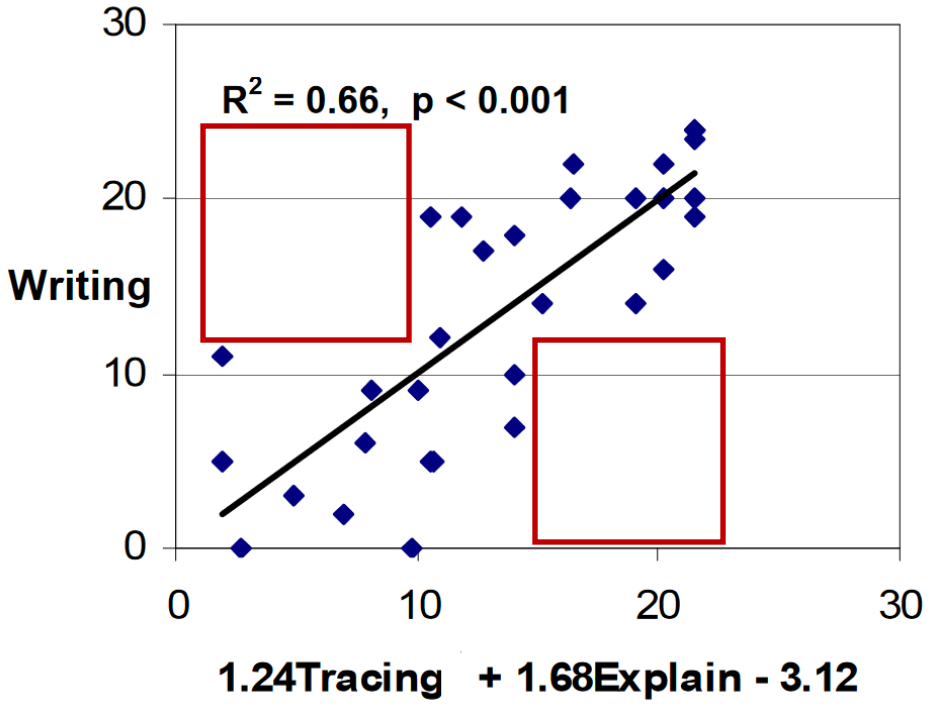
Die Funktion prüft, ob eine Liste aufsteigend sortiert ist.

Lister, R. (2022). Some thoughts on designing eye movement studies for novice programmers.

Zusammenhang zwischen Code Tracing, Explaining und Writing



Lopez et al., 2008



Venables et al., 2009

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming.
Venables, A., Tan, G., & Lister, R. (2009). A Closer Look at Tracing, Explaining and Code Writing Skills in the Novice Programmer.

Programmcode lesen, erklären und schreiben

- Lister (2022) geht von einer gestuften Entwicklung der Programmierfähigkeit bei Programmieranfänger:innen (Novizen) aus:
  - Stufe 1: Pre-Tracing
    - Noch nicht in der Lage, Code zuverlässig zu tracen
    - Grund dafür sind in erster Linie Fehlvorstellungen bezüglich der Notional Machine
  - Stufe 2: Tracing (induktiv)
    - In der Lage, Code zuverlässig zu tracen
    - Erschließt sich die Bedeutung des Codes ausschließlich induktiv; d.h. basierend auf dem Vergleich der Belegung der Variablen vor und nach der Ausführung des Programms
  - Stufe 3: Post-Tracing (deduktiv)
    - Ist in der Lage deduktiv über den Code nachzudenken
    - Kann sich die Bedeutung des Codes allein durch Lesen erklären
    - Entwickelt einen kohärenten, zielgerichteten Ansatz zum Schreiben von Code

Lister, R. (2022). Some thoughts on designing eye movement studies for novice programmers.

Erkenntnisse über das Leseverhalten

- Scan-Phase: Etwa 70% des Quellcodes wird in den ersten 30% der Zeit erfasst *(Uwano et al., 2006)*
- Programmierer:innen, die weniger Zeit für das anfängliche Scannen des Codes aufwenden, brauchen tendenziell mehr Zeit, um Fehler zu finden *(Sharif et al., 2012)*
- Kommentare helfen beim "Chunken" von größeren Code-Blöcken *(Fan, 2010)*
- Im Vergleich zum Lesen von natürlich-sprachlichen Texten längere Fixationen und höhere Anzahl an Rücksprüngen *(Busjahn et al. 2011)*
  - Das Lesen von Programmcode ist weniger linear als das Lesen natürlichsprachlicher Texte
- Wiederkehrende Muster in den Augenbewegungen häufiger bei Programmieranfänger:innen (Novizen) zu beobachten *(Bednarik, 2012)*
- Wahl der Programmiersprache hat Einfluss darauf, wie Probleme angegangen und gelöst werden *(Turner et al., 2014)*

Uwano, H.; Nakamura, M.; Monden, A. & Matsumoto, K. (2006). Analyzing Individual Performance of Source Code Review Using Reviewers' Eye Movement.
Fan, Q. (2010). The Effects of Beacons, Comments, and Tasks on Program Comprehension Process in Software Maintenance.
Busjahn, T.; Schulte, C. & Busjahn, A. (2011): Analysis of code reading to gain more insight in program comprehension.
Bednarik, R. (2012). Expertise-Dependent Visual Attention Strategies Develop over Time during Debugging with Multiple Code Representations.

Programmverstehen

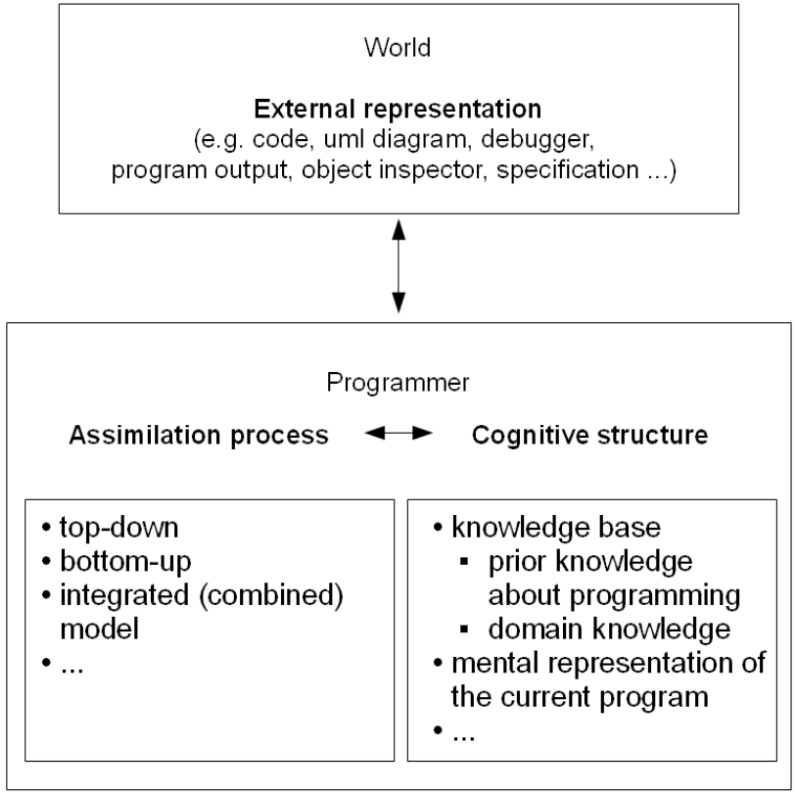
Modell des Programmverstehens

Blockmodell

Modell des Programmverstehens

(Schulte et al., 2010)

- Programmverstehen
  - Prozess, bei dem eine Person ihre eigene mentale Repräsentation des Programms konstruiert
- Externale Repräsentation
  - Darstellung des Programms (Source code, UML-Diagramm, Syntax-Hervorhebung, …)
- Kognitive Struktur
  - Vorwissen (Wissensbasis)
  - Mentale Repräsentation des Programms
- Assimilationsprozess
  - Aufbau einer mentalen Repräsentation unter Verwendung der Wissensbasis und der Repräsentation des Programms



Primäre Komponenten des Programmverstehens *(Schulte et al., 2010)*

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

Macro structure	Understanding the overall structure of the program text	Understanding the "algorithm" of the program	Understanding the goal/ the purpose of the program (in its context)
Relations	References between blocks, e.g.: method calls, object creation, accessing data, ...	Sequence of method calls "object sequence diagrams"	Understanding how subgoals are related to goals, how function is achieved by subfunctions
Blocks	'Regions of Interests' (ROI) that syntactically or semantically build a unit	Operation of a block, a method, or a ROI (as sequence of statements)	Function of a block, maybe seen as subgoal
Atoms	Language elements	Operation of a statement	Function of a statement. Goal only understandable in context
	Text surface	**Program execution (data flow and control flow)**	**Functions (as means or as purpose), goals of the program**
Duality	***Structure***		***Function***

Blockmodell

(Schulte et al., 2010)

- Dualität: Struktur und Funktion *(siehe Kapitel 2)*
- Lernpfade
  - Das Modell kann in unterschiedlicher Reihenfolge durchlaufen werden
    - Bottom-Up
    - Top-Down

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

Notional Machines (NM)

(Boulay, 1986; Sorva, 2013; Fincher et al., 2020)

Notional Machines

(Fincher et al., 2020)

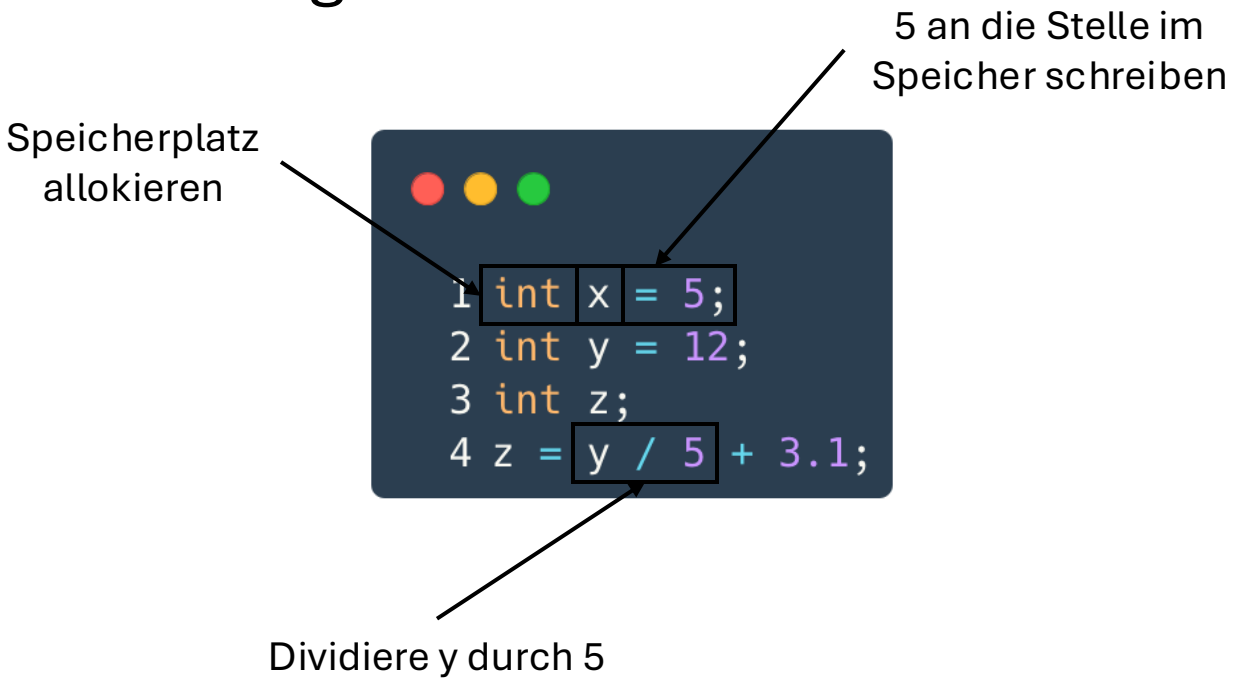
- Notional Machine (NM)
  - Abstraktes, vereinfachtes Modell der Funktionsweise eines Computers hinsichtlich der Ausführung eines Programms
  - Pädagogisches Hilfsmittel
- Eine Notional Machine …
  - hat einen **pädagogischen Zweck**,
  - ihre allgemeine **Funktion** ist es, die Aufmerksamkeit auf einen verborgenen (nicht offensichtlichen) Aspekt von Programmen oder Computern zu lenken oder diesen hervorzuheben
  - hat einen **Fokus** auf einen bestimmten Aspekt von Programmen
  - nimmt eine bestimmte **Darstellung** an, die bestimmte Aspekte dieses Schwerpunkts hervorhebt.

Fincher, S., Jeuring, J., Miller, C. S., Donaldson, P., du Boulay, B., Hauswirth, M., Hellas, A., Hermans, F., Lewis, C., Mühling, A., Pearce, J. L., & Petersen, A. (2020). Notional Machines in Computing Education: The Education of Attention.

Beispiel

(Bruce-Lockhart and Norvell, 2007)

- Betrachten Sie das folgende Programm in C:



- Aus wie vielen (atomaren) Anweisungen setzt sich das Programm zusammen?
- Wie wird das Programm vom Computer interpretiert und ausgeführt?

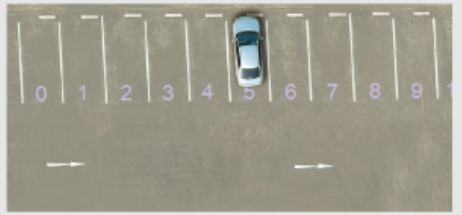
Bruce-Lockhart, M. P. & Norvell, T. S. (2007). Developing Mental Models of Computer Programming Interactively Via the Web.

Konzept in der Programmiersprache (PL)

Analogie in der Notional Machine (NM)

Array as Row of Spaces in Parking Lot

Programming Paradigm:
imperative
Programming Language:
any



Conceptual Advantage
Builds from the notion of a variable as a parking space and leverages the notion that spaces in larger parking lots are often numbered.

PL	NM
array	row of cars in a parking lot
array index	space number in lot
array element	car
array value	specific car in specific space

Form
Analogy

Draws Attention To
The use of indices in arrays as well as the array's construction from contiguous adjacent variables in memory.

Cost
Short time to introduce but learners need to be familiar with parking lots and how someone might locate a specific car in a row in a parking lot.

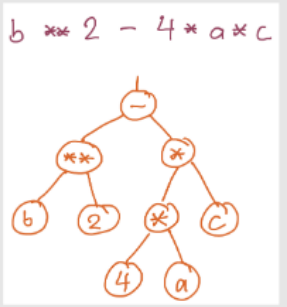
Notes/Other
If in a statically-typed language one can discuss rows that are for cars only, trucks only, motorcycles only, etc.

Origin/Source
Own practice

Attribution
Used by Jan, collected by ***Jan***

Bruce-Lockhart, M. P. & Norvell, T. S. (2007). Developing Mental Models of Computer Programming Interactively Via the Web.

Expression as Tree



Programming Paradigm:
any
Programming Language:
any

Conceptual Advantage
Makes explicit the tree structure of expressions, which otherwise is quite hidden in many text-based languages

PL	NM
expression	tree
operator	parent node
operand	child node

Form
Handmade representation

Draws Attention To
Expression structure, how evaluation proceeds, how types are determined

Use When
Distinguishing between expressions and statements, how to determine the value of an expression (and explaining associativity, precedence)

Cost
Learn a visual representation

Notes/Other
High school teachers were excited about this, mentioned this should also be used in math

Origin/Source
Own practice
(inspired by compiler ASTs)

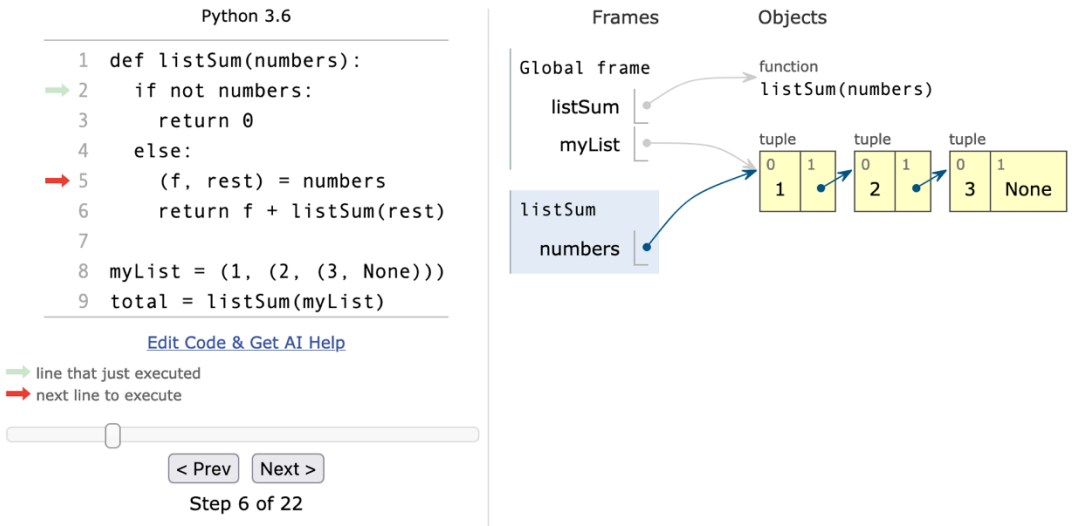
Attribution
Used by Matthias; collected by ***Matthias***

Bruce-Lockhart, M. P. & Norvell, T. S. (2007). Developing Mental Models of Computer Programming Interactively Via the Web.

Python Tutor (Link)

- Visueller Debugger
- Programm wird Schritt-für-Schritt durchlaufen
- Rücksprünge möglich
- Unterstützte Sprachen: Python, Java, C, C++ und JavaScript

Ziel: Abbau von Fehlvorstellungen bzgl. der Notional Machine



Python 3.6

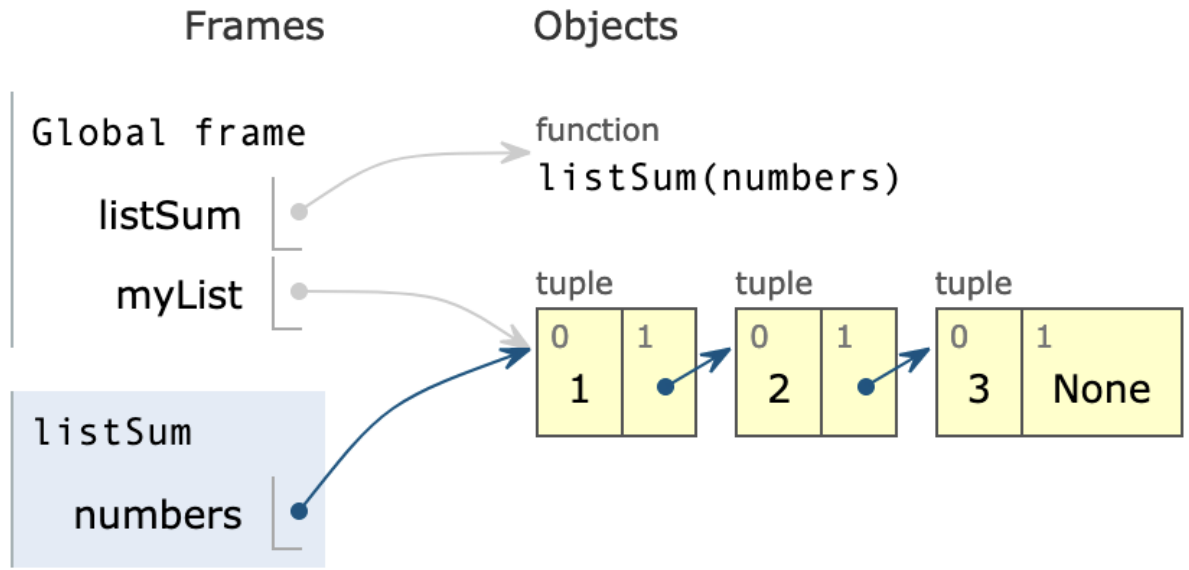
```
1  def listSum(numbers):
2    if not numbers:
3      return 0
4    else:
5      (f, rest) = numbers
6      return f + listSum(rest)
7
8  myList = (1, (2, (3, None)))
9  total = listSum(myList)
```

Edit Code & Get AI Help

line that just executed
next line to execute

< Prev Next >

Step 6 of 22



Python 3.6

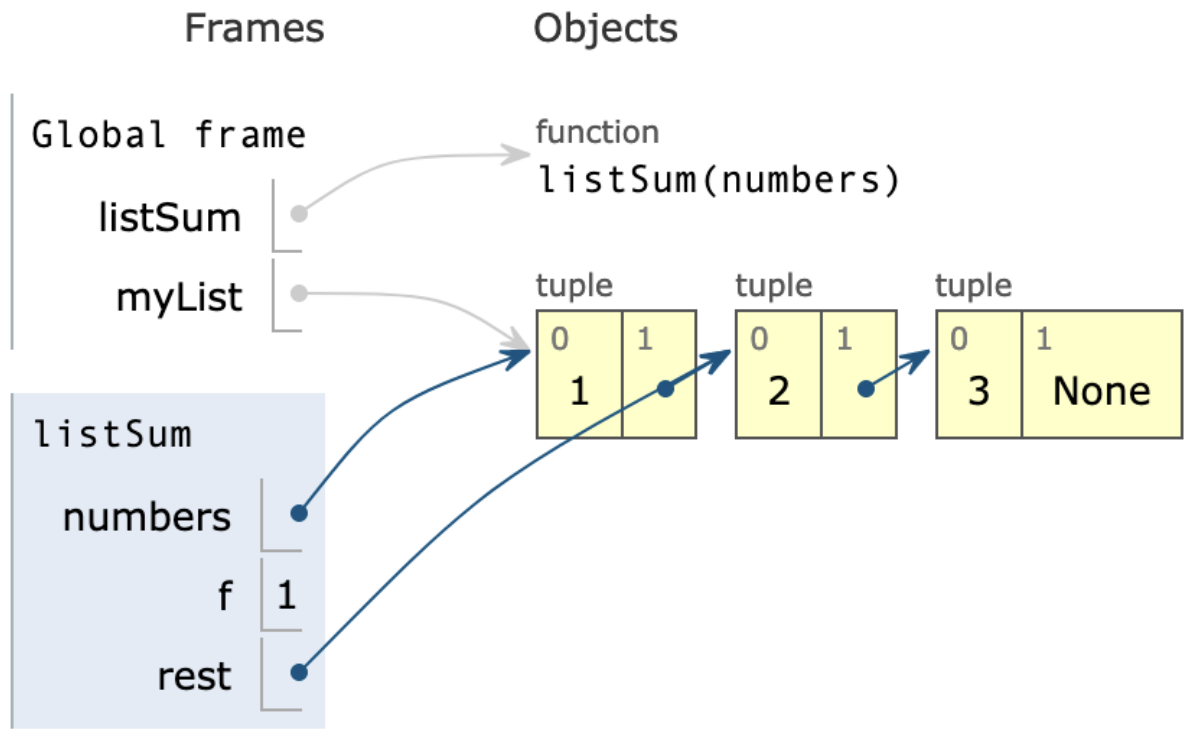
```
1  def listSum(numbers):
2    if not numbers:
3      return 0
4    else:
5      (f, rest) = numbers
6      return f + listSum(rest)
7
8  myList = (1, (2, (3, None)))
9  total = listSum(myList)
```

Edit Code & Get AI Help

line that just executed
next line to execute

< Prev Next >

Step 7 of 22



Python 3.6

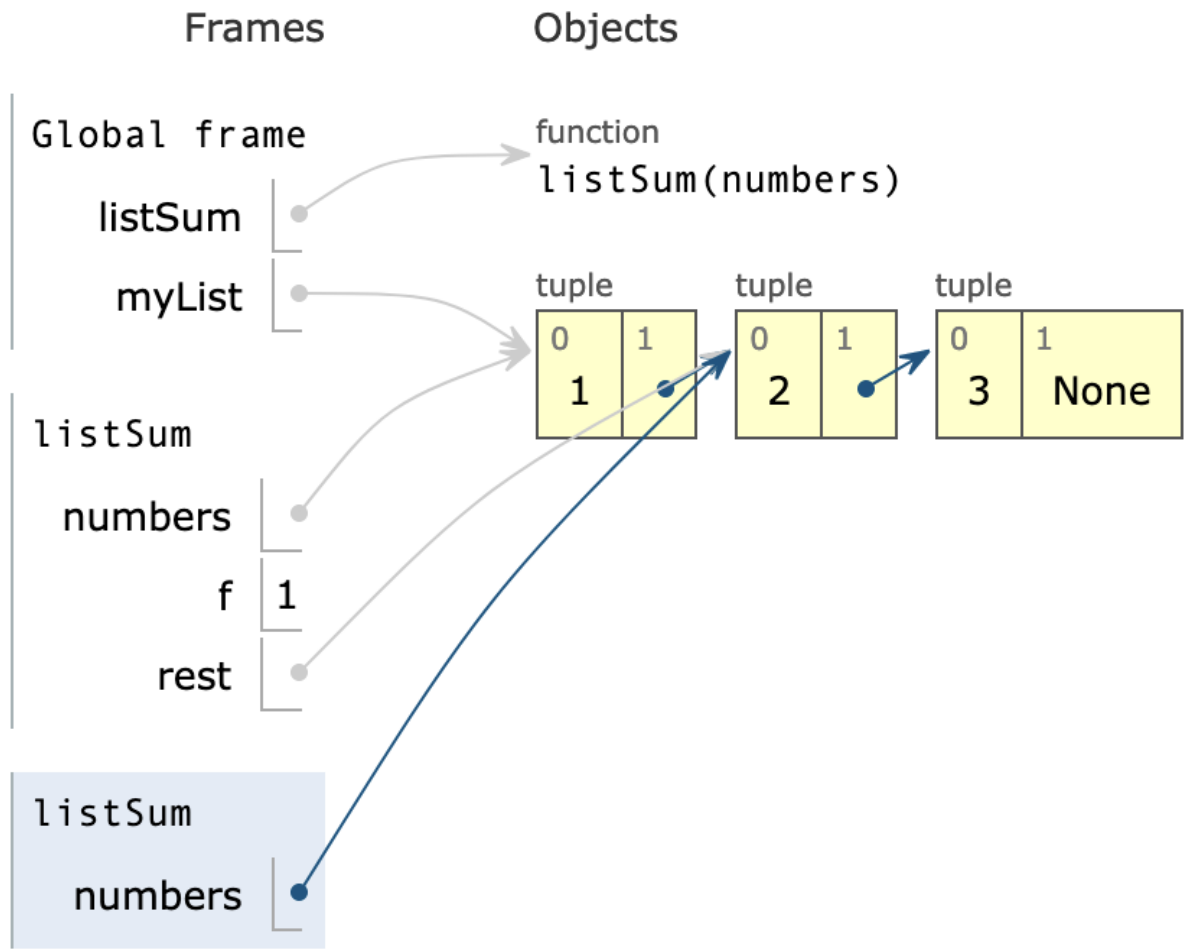
```
1 def listSum(numbers):
2   if not numbers:
3     return 0
4   else:
5     (f, rest) = numbers
6     return f + listSum(rest)
7
8 myList = (1, (2, (3, None)))
9 total = listSum(myList)
```

Edit Code & Get AI Help

line that just executed
next line to execute

< Prev Next >

Step 8 of 22



Referenzen

- Fan, Q. (2010). The effects of beacons, comments,
and tasks on program comprehension process in software maintenance. University of Maryland, Baltimore County.
- Fincher, S., Jeuring, J., Miller, C. S., Donaldson, P., du Boulay, B., Hauswirth, M., Hellas, A., Hermans, F., Lewis, C., Mühling, A., Pearce, J. L., & Petersen, A. (2020). Notional Machines in Computing Education: The Education of Attention. In ITiCSE-WGR '20: Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (S. 21 – 50). Trondheim: ACM. https://doi.org/10.1145/3437800.3439202.
- Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., & Thomas, L. (2004). A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *SIGCSE Bull.*, *36*(4), 119–150. https://doi.org/10.1145/1041624.1041673.

Referenzen

- Baldwin, L. P. & Kuljis, J. (2000). Visualisation techniques for learning and teaching programming. In ITI 2000. Proceedings of the 22nd International Conference on Information Technology Interfaces (S. 83 – 90). https://doi.org/10.2498/cit.2000.04.03.
- Bednarik, R. (2012). Expertise-Dependent Visual Attention Strategies Develop over Time during Debugging with Multiple Code Representatio ns. In International Journal of Human-Computer Studies 70 (2) (143 – 155). https://doi.org/10.1016/j.ijhcs.2011.09.003.
- Bruce-Lockhart, M. P., & Norvell, T. S. (2007). Developing Mental Models of Computer Programming Interactively Via the Web. 2007 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, S3H-3-S3H-8. https://doi.org/10.1109/FIE.2007.4418051.

Referenzen

- Busjahn, T.; Schulte, C. & Busjahn, A. (2011): Analysis of code reading to gain more insight in program comprehension. In Koli Calling '11: Proceedings of the 11th Koli Calling International Conference on Computing Education Research (S. 1 – 9). Koli: ACM. https://doi.org/10.1145/2094131.2094133.
- Du Boulay, B. (1986). Some difficulties of learning to program. Journal of Educational Computing Research, 2(1), 57–73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9.
- Duran, R.; Zavgorodniaia, A. & Sorva, J. (2022). Cognitive Load Theory in Computing Education Research. In ACM Transactions on Computing Education 22 (4) (S. 1 – 27). New York: ACM. https://doi.org/10.1145/3483843.

Referenzen

- Lister, R. (2022). Some thoughts on designing eye movement studies for novice programmers. In 2022 IEEE/ACM 10th International Workshop on Eye Movements in Programming (EMIP) (S. 15 – 22). Pittsburgh: ACM. https://doi.org/10.1145/3524488.3527363.
- Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In ICER '08: Proceedings of the Fourth international Workshop on Computing Education Research (S. 101 – 112). Santa Barbara : ACM. https://doi.org/10.1145/1404520.1404531.
- Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., & Paterson, J. H. (2010). An introduction to program comprehension for computer science educators. In ITiCSE-WGR '10: Proceedings of the 2010 ITiCSE working group reports (S. 65 – 86). Ankara: ACM. https://doi.org/10.1145/1971681.1971687.

Referenzen

- Sharif, B.; Falcone, M. & Maletic, J. I. (2012). An Eye-Tracking Study on the Role of Scan Time in Finding Source Code Defects. In ETRA '12: *Proceedings of the Symposium on Eye Tracking Research and Applications* (S. 381 – 384). Santa Barbara: ACM. https://doi.org/10.1145/2168556.2168642.
- Sorva, J. (2013). Notional machines and introductory programming education. In ACM Transactions on Computing Education 13 (2) (S. 1 – 31). New York: ACM. https://doi.org/10.1145/2483710.2483713.
- Sweller, J.; Van Merrienboer, J. J. & Paas, F. G. (1998). Cognitive architecture and instructional design. In Educational Psychology Review 10 (S. 251 – 296).

Referenzen

- Turner, R., Falcone, M., Sharif, B., & Lazar, A. (2014). An Eye-Tracking Study Assessing the Comprehension of C++ and Python Source Code. Proceedings of the Symposium on Eye Tracking Research and Applications, 231–234. https://doi.org/10.1145/2578153.2578218
- Uwano, H.; Nakamura, M.; Monden, A. & Matsumoto, K. (2006). Analyzing Individual Performance of Source Code Review Using Reviewers' Eye Movement. In ETRA '06: *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications* (S. 133 – 140). San Diego: ACM Press. https://doi.org/10.1145/1117309.1117357.
- Venables, A., Tan, G., & Lister, R. (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. In: ICER '09: Proceedings of the fifth international workshop on Computing education research workshop (S. 117 – 128). Berkley: ACM. https://doi.org/10.1145/1584322.1584336.

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