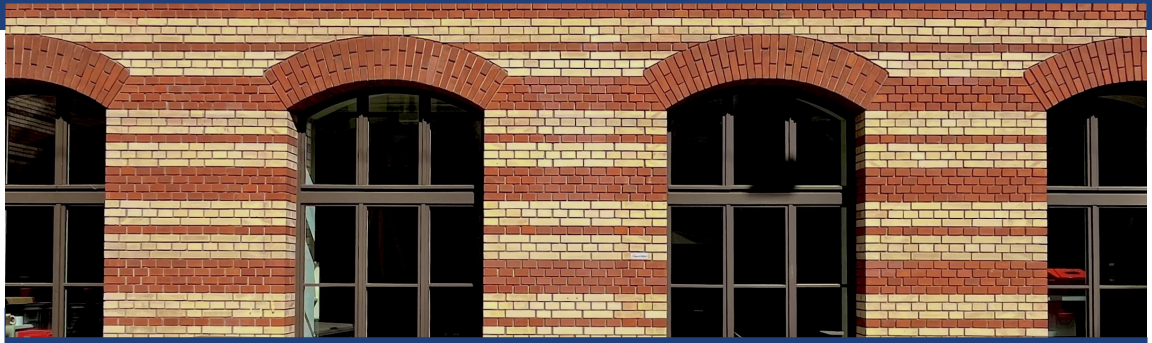


Übungsstunde — Informatik — 08

Adel Gavranović

Rekursion, Structs

Übersicht

Follow-up
- `const` und Funktionen
- Aufgabe "Towers of Hanoi"

Structs
- Aufgabe "Geometry Exercise"

Rekursion
- Aufgabe "Power Set"

`n.ethz.ch/~agavranovic`

Material

Webpage

Mail

1. Follow-up

Follow-up aus letzter Übungsstunde

Follow-up aus letzter Übungsstunde

- Applausreform

Follow-up aus letzter Übungsstunde

- Applausreform
- Gute Nachrichten (und Lob)

Follow-up aus letzter Übungsstunde

- Applausreform
- Gute Nachrichten (und Lob)
- `const` und Funktionen
- Towers of Hanoi

1. Follow-up

1.1. `const` und Funktionen

`const` und Funktionen

const und Funktionen

1.
```
void Funktion(const Type& n){                    //  const als Argument
    // n kann nicht verändert werden
} // sehr oft sinnvoll; muss man kennen!
```

const und Funktionen

1.
```
void Funktion(const Type& n){                    //  const als Argument
    // n kann nicht verändert werden
} // sehr oft sinnvoll; muss man kennen!
```

2.
```
const Type& Funktion(const Type& n){             //  const als Return
    // n kann nicht verändert werden
    return n; // ursprüngliches n wird returned
} // manchmal sinnvoll; ziemlich spezifish
```

const und Funktionen

1.
```
void Funktion(const Type& n){                    //  const als Argument
    // n kann nicht verändert werden
} // sehr oft sinnvoll; muss man kennen!
```

2.
```
const Type& Funktion(const Type& n){             //  const als Return
    // n kann nicht verändert werden
    return n; // ursprüngliches n wird returned
} // manchmal sinnvoll; ziemlich spezifish
```

3.
```
void Funktion(Type n) const {                    // const-Funktionen
    // n kann verändert werden
    // nichts in der Klasse, der die Funktion angehört,
    // kann verändert werden. [Klassen wurden noch nicht behandelt]
} // oft sinnvoll; wird vielleict besprochen im Thema "Klassen"
```

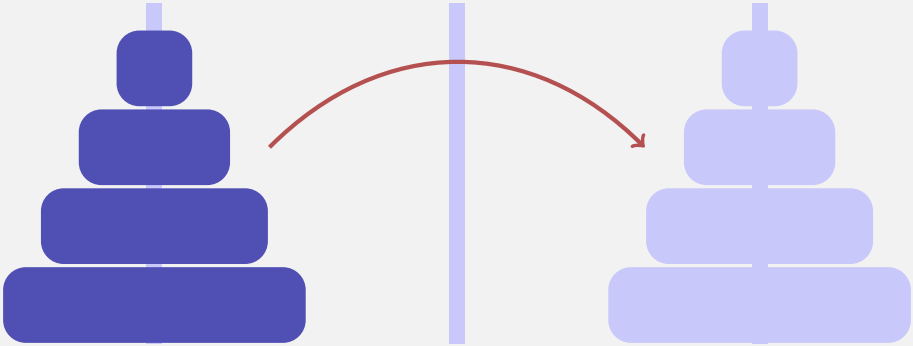
1. Follow-up

1.2. Aufgabe "Towers of Hanoi"

Experiment: Die Türme von Hanoi

Links Mitte Rechts

Experiment: Die Türme von Hanoi

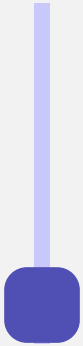


Die Türme von Hanoi - So gehts!

Links Mitte Rechts

Die Türme von Hanoi - So gehts!

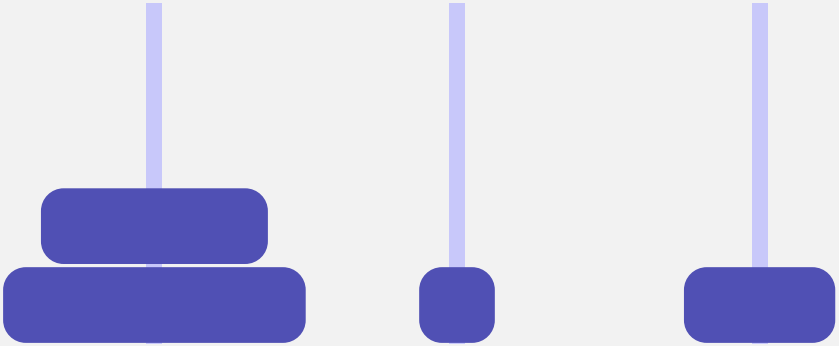
Links



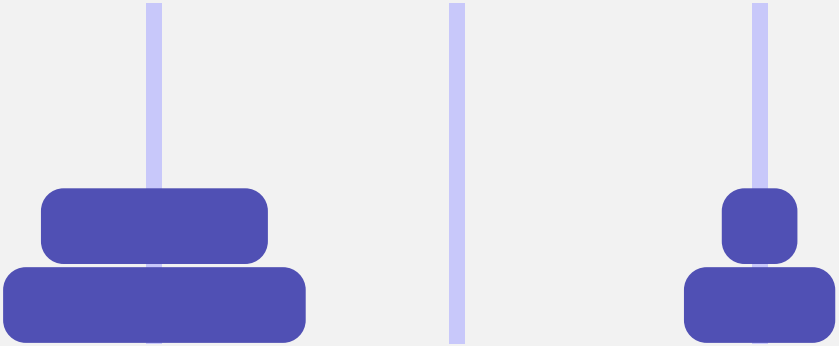
Mitte

Rechts

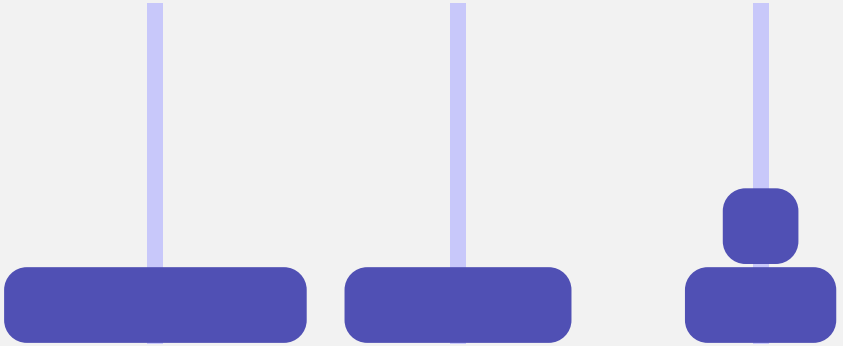
Die Türme von Hanoi - So gehts!



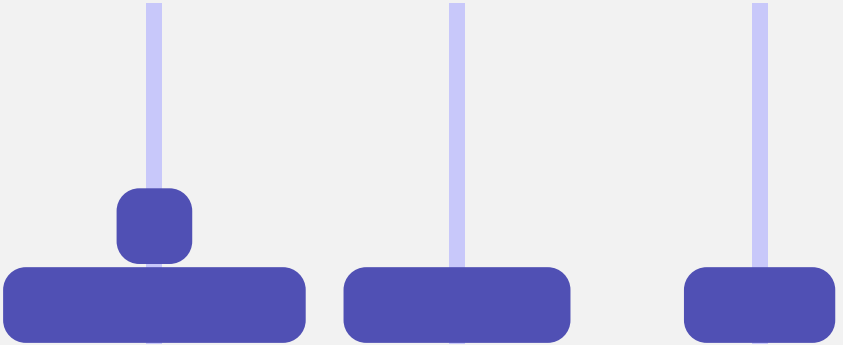
Die Türme von Hanoi - So gehts!



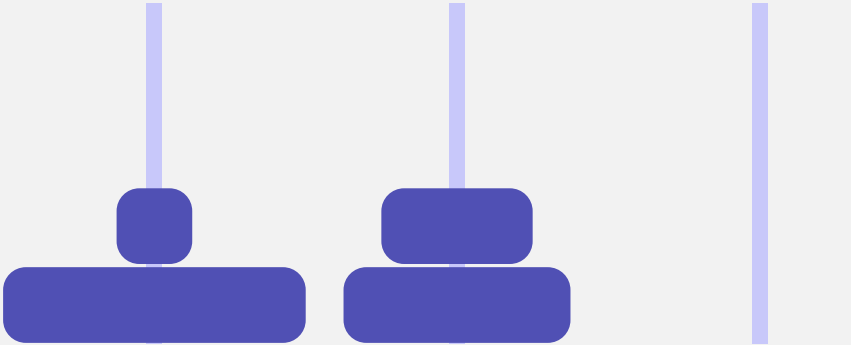
Die Türme von Hanoi - So gehts!



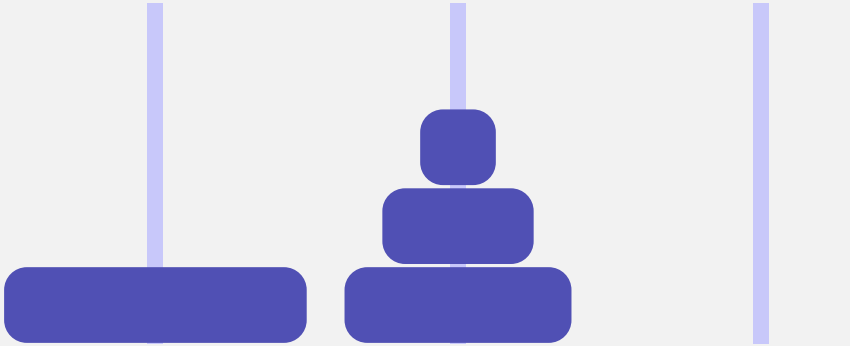
Die Türme von Hanoi - So gehts!



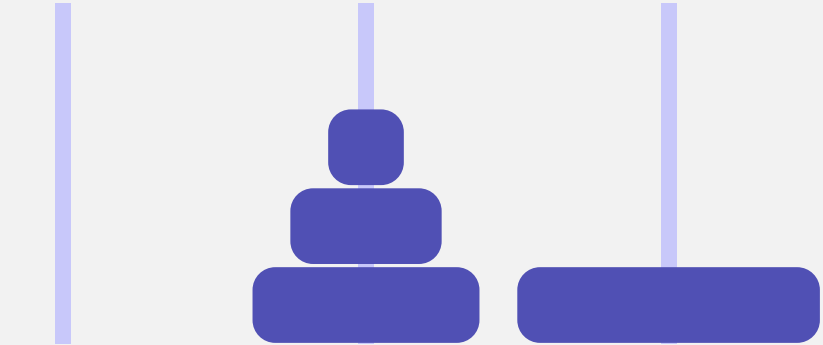
Die Türme von Hanoi - So gehts!



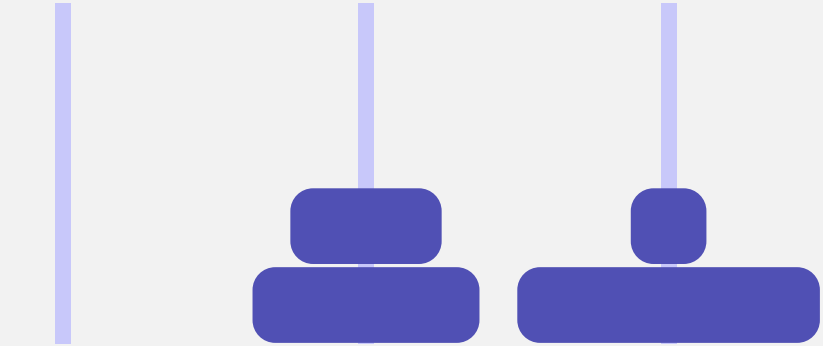
Die Türme von Hanoi - So gehts!



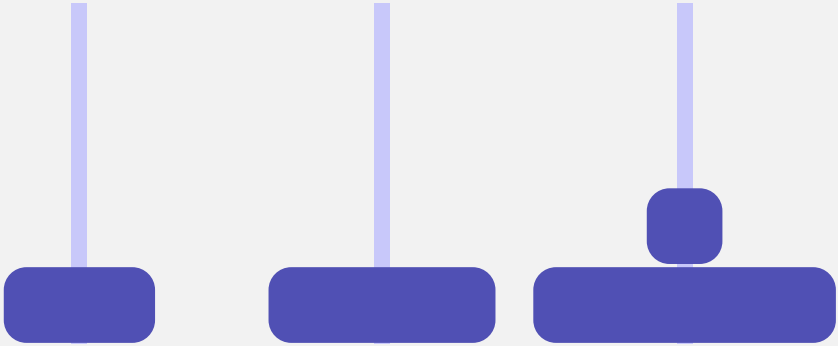
Die Türme von Hanoi - So gehts!



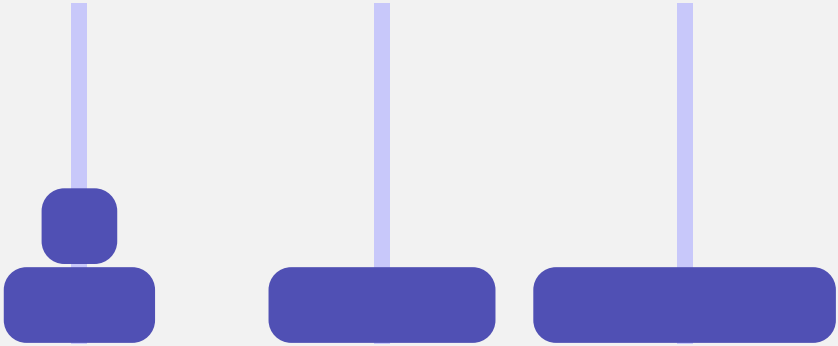
Die Türme von Hanoi - So gehts!



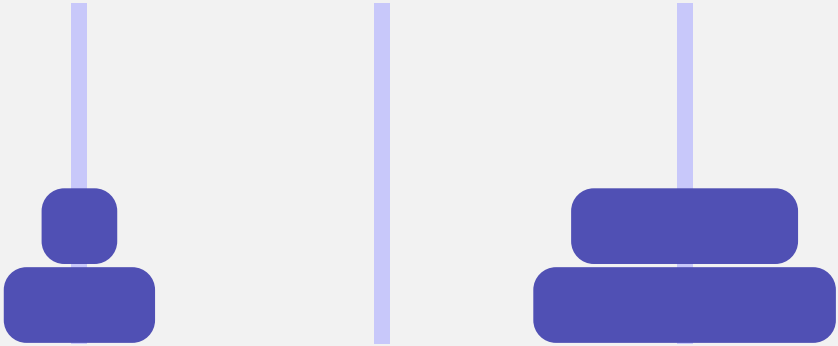
Die Türme von Hanoi - So gehts!



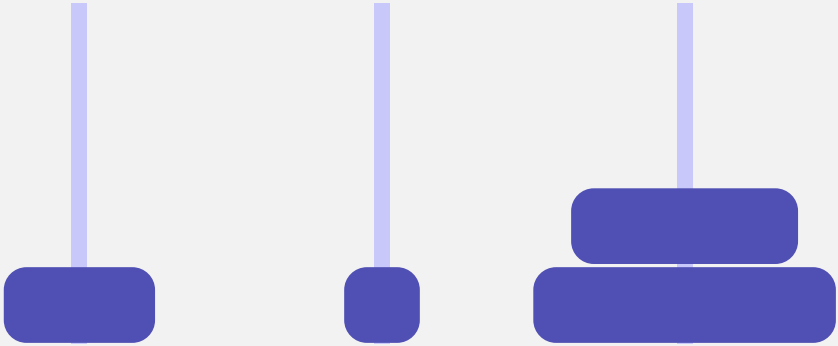
Die Türme von Hanoi - So gehts!



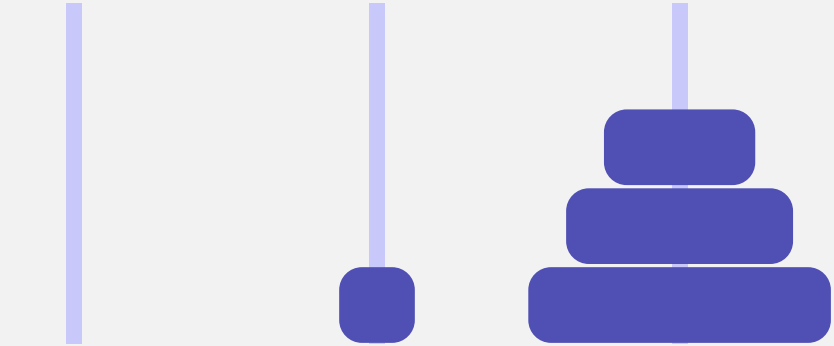
Die Türme von Hanoi - So gehts!



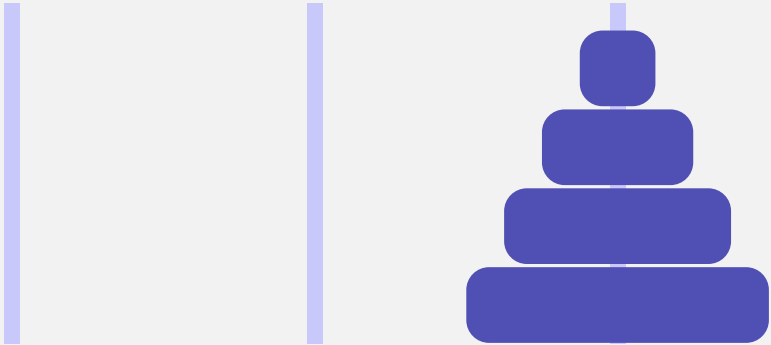
Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!

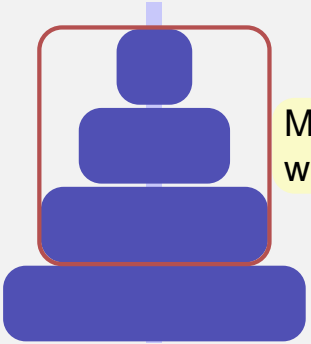


Links Mitte Rechts

Die Türme von Hanoi - Rekursiver Lösungsansatz

Links Mitte Rechts

Die Türme von Hanoi - Rekursiver Lösungsansatz

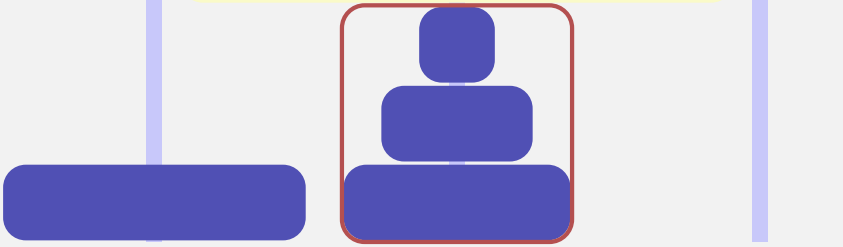


Mal *angenommen*, wir wüssten wie man ...

Links Mitte Rechts

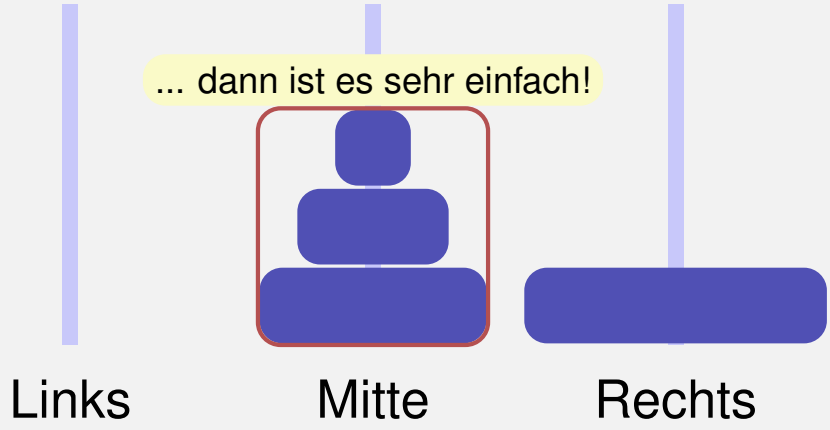
Die Türme von Hanoi - Rekursiver Lösungsansatz

... *drei* Scheiben von einem Stapel zum anderen bringt ...

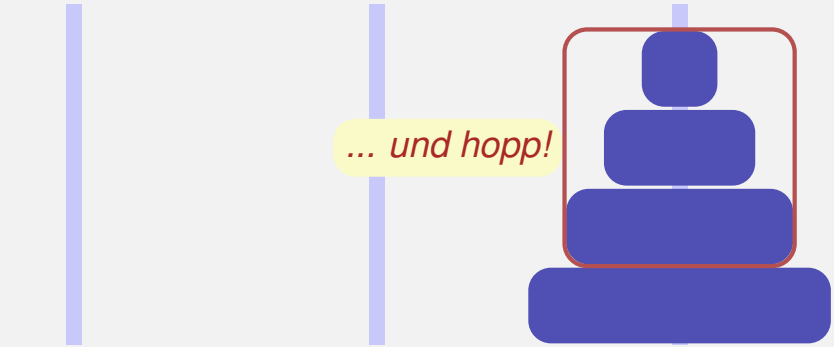


Links Mitte Rechts

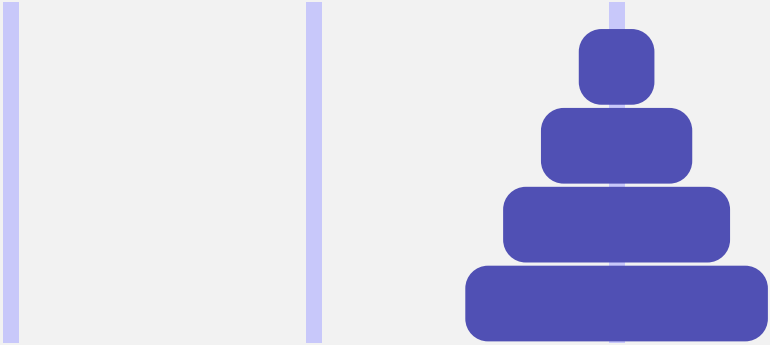
Die Türme von Hanoi - Rekursiver Lösungsansatz



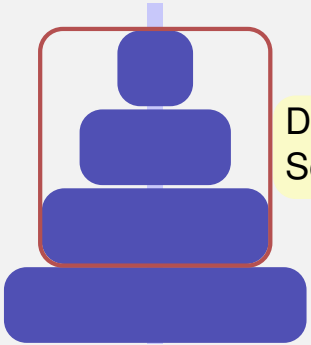
Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz



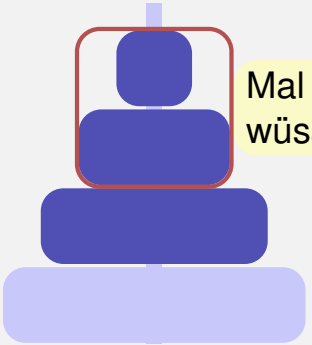
Die Türme von Hanoi - Rekursiver Lösungsansatz



Doch *wie* können wir drei Scheiben bewegen?

Links Mitte Rechts

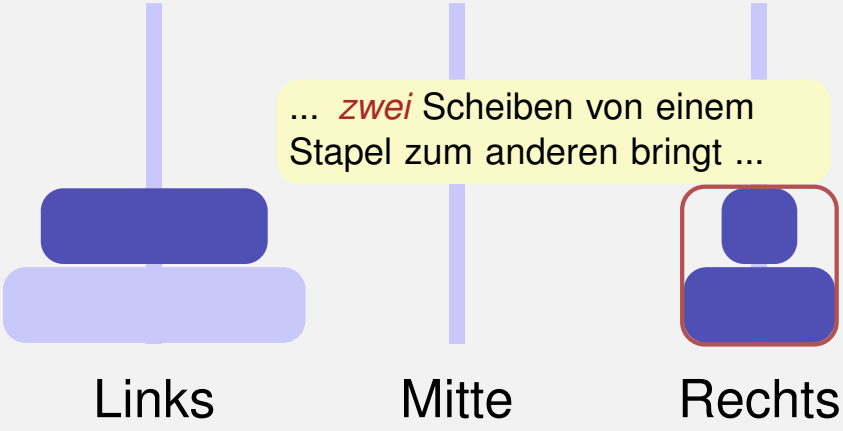
Die Türme von Hanoi - Rekursiver Lösungsansatz



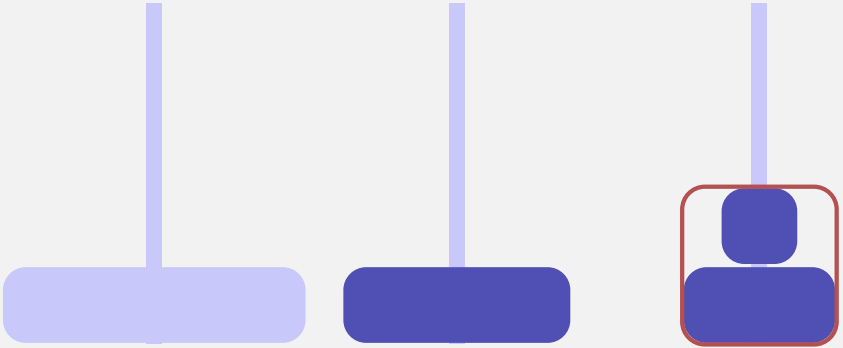
Mal *angenommen*, wir wüssten wie man ...

Links Mitte Rechts

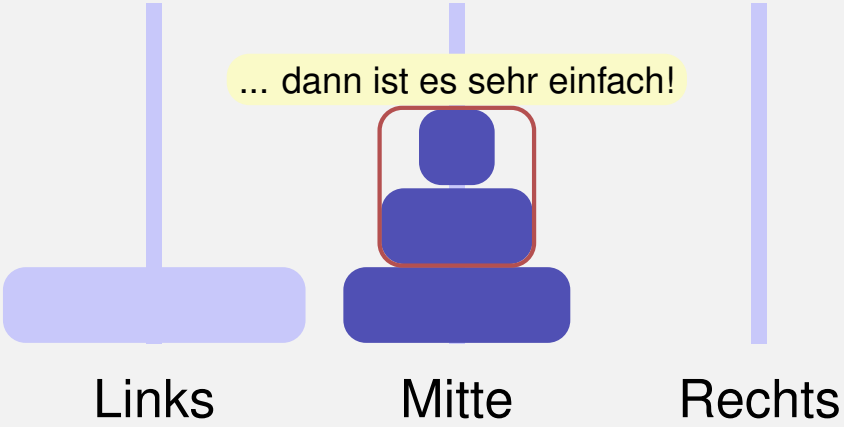
Die Türme von Hanoi - Rekursiver Lösungsansatz



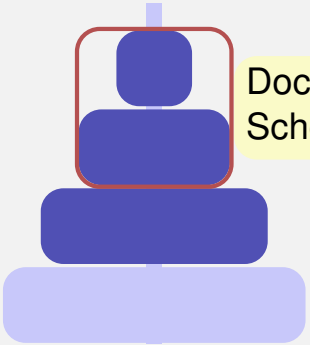
Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz

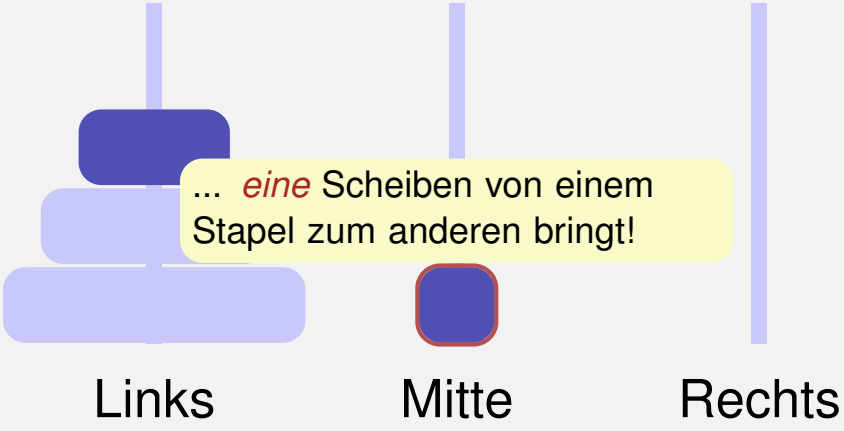


Doch *wie* können wir zwei Scheiben bewegen?

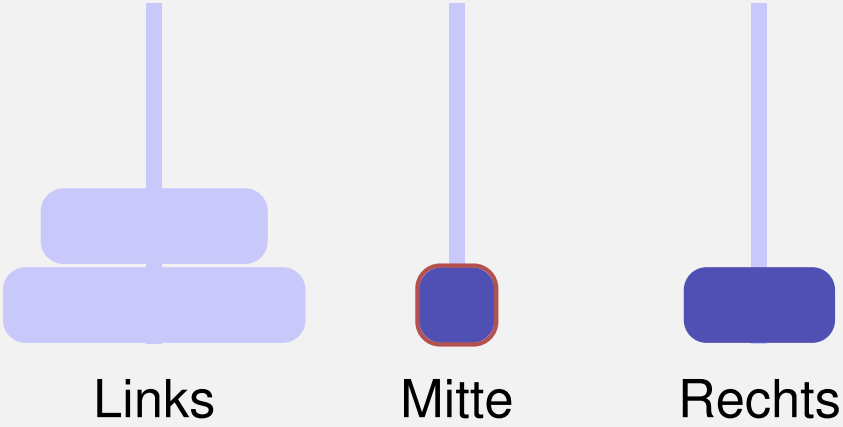
Links Mitte Rechts

Die Türme von Hanoi - Rekursiver Lösungsansatz

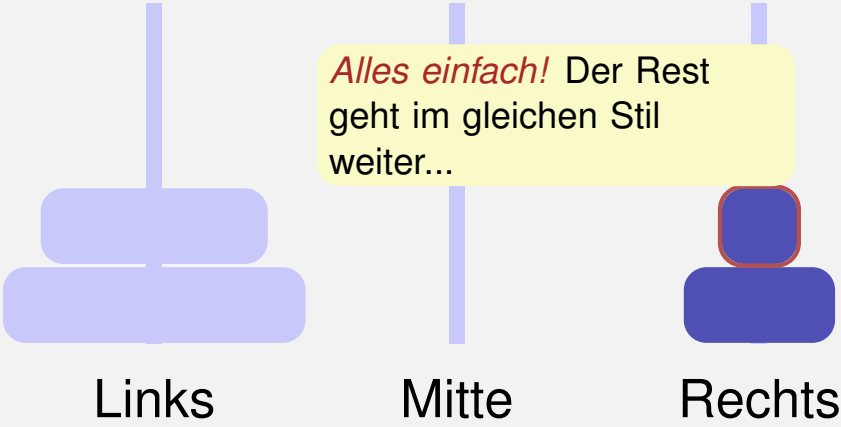
Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz



Die Türme von Hanoi - Rekursiver Lösungsansatz



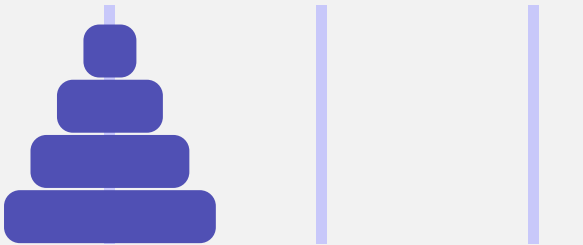
Aufgabe Towers of Hanoi

- Öffnet "Towers of Hanoi" auf **code** expert

Aufgabe Towers of Hanoi

- Öffnet "Towers of Hanoi" auf **code** expert
- Programmiert eine Lösung

Die Türme von Hanoi - Code



Bewege 4 Scheiben von `left` nach `right` mit Hilfsstapel `middle`:

```
move(4,"left","middle","right")
```

Die Türme von Hanoi - Code

move(n**,** src**,** aux**,** dst**)** $\Rightarrow$

1. Bewege die obersten $n-1$ Scheiben von src nach aux mit Hilfsstapel dst:
   move($n-1$**,** src**,** dst**,** aux**);**
2. Bewege 1 Scheibe von src nach dst
   move(1, src**,** aux**,** dst**);**
3. Bewege die obersten $n-1$ Scheiben von aux nach dst mit Hilfsstapel src:
   move($n-1$**,** aux**,** src**,** dst**);**

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case



    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);


    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);

    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);
        move(n-1, aux, src, dst);
    }
}
```

Die Türme von Hanoi - Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case ('move' the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);
        move(n-1, aux, src, dst);
    }
}
int main() {
    move(4, "left", "middle", "right");
    return 0;
}
```

Die Türme von Hanoi - Code Alternative

```
void move(int n, const string &src, const string &aux, const string &dst){
    // base case
    if (n == 0) return;

    // recursive case
    move(n-1, src, dst, aux);
    std::cout << src << " --> " << dst << "\n";
    move(n-1, aux, src, dst);
}


int main() {
    move(4, "left", "middle", "right");
    return 0;
}
```

Fragen/Unklarheiten?

2. Feedback zu **code** expert

Allgemeines bezüglich **code** expert

Ein paar Vereinfachungen für euren Code[1]

```
if(condition == true){
  // ...
}
```

→

```
if(condition){
  //...
}
```

```
if(condition){
  return true;
} else {
  return false;
}
```

→

```
return condition;
```

in Funktionen, die einen `bool` zurückgeben

[1] Nicht vergessen: Vereinfachungen sind nicht immer besser für die Verständlichkeit

Allgemeines bezüglich **code** expert

Allgemeines bezüglich **code** expert

```
if (condition) {
    // thing
}
else {                  // PFUUUUIII
    // other thing
}
```

Allgemeines bezüglich **code** expert

```
if (condition) {
    // thing
}
else {                    // PFUUUUIII
    // other thing
}
```

Setzt die Klammern nach **else** richtig. Wir programmieren hier kein Python!

```
if (condition) {
    // thing
} else {
    // other thing
}
```

Aufgabe "Pre- and post-conditions"

Aufgabe "Pre- and post-conditions"

```
// PRE n is an integer and larger than 0
// POST gives amount of digits of n
```

Aufgabe "Pre- and post-conditions"

```
// PRE n is an integer and larger than 0
// POST gives amount of digits of n
```

Aligned, präziser, einfacher zu lesen:

```
// PRE:  n > 0
// POST: returns number of digits of n (in decmimal representation)
```

Aufgabe "Fixing Functions"

Aufgabe "Fixing Functions"

- Uninitialisierte Variabeln haben einen unbestimmbaren Wert (oft 0, aber eben nicht immer!)

Aufgabe "Fixing Functions"

- Uninitialisierte Variabeln haben einen unbestimmbaren Wert (oft 0, aber eben nicht immer!)
- Benutzt `asserts` um PREs einzuhalten

Wichtige Änderung bezüglich Feedback

Wichtige Änderung bezüglich Feedback

- Aufgrund zu hoher Auslastung, bekommt man ab heute Feedback nur noch auf Anfrage (ausser ich sehe gerade etwas grundlegend falsches)

Wichtige Änderung bezüglich Feedback

- Aufgrund zu hoher Auslastung, bekommt man ab heute Feedback nur noch auf Anfrage (ausser ich sehe gerade etwas grundlegend falsches)
- Diese "Anfrage" kann folgendermassen aussehen und ist gaaaaanz oben im Code anzubringen

```
// FEEDBACK PLEASE
// - especially regarding lines 12, 13 and 42
// QUESTIONS
// - [re: line 42] I was wondering if [...]
```

Wichtige Änderung bezüglich Feedback

- Aufgrund zu hoher Auslastung, bekommt man ab heute Feedback nur noch auf Anfrage (ausser ich sehe gerade etwas grundlegend falsches)
- Diese "Anfrage" kann folgendermassen aussehen und ist gaaaaanz oben im Code anzubringen

```
// FEEDBACK PLEASE
// - especially regarding lines 12, 13 and 42
// QUESTIONS
// - [re: line 42] I was wondering if [...]
```

- TA-Punkte werden noch immer verteilt

Wichtige Änderung bezüglich Feedback

- Aufgrund zu hoher Auslastung, bekommt man ab heute Feedback nur noch auf Anfrage (ausser ich sehe gerade etwas grundlegend falsches)
- Diese "Anfrage" kann folgendermassen aussehen und ist gaaaaanz oben im Code anzubringen

```
// FEEDBACK PLEASE
// - especially regarding lines 12, 13 and 42
// QUESTIONS
// - [re: line 42] I was wondering if [...]
```

- TA-Punkte werden noch immer verteilt
- Falls irgendwo die XP auf 0 gesetzt werden müssen, wird im Feedback erwähnt sein weshalb

Fragen bezüglich **code** expert eurerseits?

3. Lernziele

Ziele für Heute

Lernziele

Ziele für Heute

Lernziele

- ☐ Code, der `struct`s enthält schreiben und verstehen können
- ☐ Code, der Rekursion enthält schreiben und verstehen können

Ziele für Heute

Lernziele

- ☐ Code, der `struct`s enthält schreiben und verstehen können
- ☐ Code, der Rekursion enthält schreiben und verstehen können
- ☐ Durch "Towers of Hanoi" leicht traumatisiert werden

4. Zusammenfassung

5. Structs

Beispiel für Structs

Beispiel für Structs

```
struct strange {
    int n;
    bool b;
    std::vector<int> a = std::vector<int>(0);
};

int main () {
    strange x = {1, true, {1,2,3}};
    strange y = x;                                  // all elements are copied
    std::cout << y.n << " " << y.a[2] << "\n";  // outputs: 1 3
    return 0;
}
```

5. Structs

5.1. Aufgabe "Geometry Exercise"

Aufgabe "Geometry Exercise"

- Öffnet "Geometry Exercise" auf **code** expert

Aufgabe "Geometry Exercise"

- Öffnet "Geometry Exercise" auf **code** expert
- Überlegt euch, wie ihr das Problem mit Stift und Papier angehen würdet

Aufgabe "Geometry Exercise"

- Öffnet "Geometry Exercise" auf **code** expert
- Überlegt euch, wie ihr das Problem mit Stift und Papier angehen würdet
- Zeit für Group Programming!

6. Rekursion

6.1. Aufgabe "Power Set"

Aufgabe "Power Set"

Rekapitulation

- Ein Potenzmenge ist die Menge aller Teilmengen

Aufgabe "Power Set"

Rekapitulation

- Ein Potenzmenge ist die Menge aller Teilmengen

$$\mathcal{P}(S) := \{X \mid X \subseteq S\}$$

Aufgabe "Power Set"

Rekapitulation

- Ein Potenzmenge ist die Menge aller Teilmengen

$$\mathcal{P}(S) := \{X \mid X \subseteq S\}$$

- Beispiel:
  - Gegeben sei die Menge $A = \{a, b, c\}$

Aufgabe "Power Set"

Rekapitulation

- Ein Potenzmenge ist die Menge aller Teilmengen

$$\mathcal{P}(S) := \{X \mid X \subseteq S\}$$

- Beispiel:
  - Gegeben sei die Menge $A = \{a, b, c\}$
  - Die Potenzmenge ist
    $\mathcal{P}(A) = \{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

Einführung zu `set.h`

- `set` ist eine selbstgebastelter Typ! (eine **`class`**)
- Wie funktioniert er? Seht selbst in `set.h`!

```
template <typename T>
class Set {
    public:
    Set(const Set& other);
    // Creates an empty set
    Set();
    // Creates a new set from a set of elements
    Set(const std::set<T>& elements);
    // Creates a new set from a single element
    Set(T element);
    // ...
};
```

Aufgabe Power Set

- Öffnet "Power Set" auf **code** expert

Aufgabe Power Set

- Öffnet "Power Set" auf **code** expert
- Überlegt euch, wie ihr das konzeptionell lösen könnt

Aufgabe Power Set

- Öffnet "Power Set" auf **code** expert
- Überlegt euch, wie ihr das konzeptionell lösen könnt
- Programmiert eine Lösung

Aufgabe Power Set

- Öffnet "Power Set" auf **code** expert
- Überlegt euch, wie ihr das konzeptionell lösen könnt
- Programmiert eine Lösung
- Tipp: Die Funktionalitäten vom Typ `set` findet ihr im `main.cpp`-file
- Mögliche Leitfragen: Für welche (einfachen) Fälle kennen wir die Lösung immer? Gibt es ein Muster, dem die Potenzmengen folgen, wenn ein weiteres Element dazukommt?

Lösung zu "Power Set" (Konzeptionell)

Gegeben: $\{a, b, c, d\}$

[2] Hier kommt der Vertrauensvorschuss der Rekursion (*Recursive Leap of Faith*) zum Tragen

Lösung zu "Power Set" (Konzeptionell)

Gegeben: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

[2] Hier kommt der Vertrauensvorschuss der Rekursion (*Recursive Leap of Faith*) zum Tragen

Lösung zu "Power Set" (Konzeptionell)

Gegeben: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

[2] Hier kommt der Vertrauensvorschuss der Rekursion (*Recursive Leap of Faith*) zum Tragen

Lösung zu "Power Set" (Konzeptionell)

Gegeben: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset
```
[2]

[2] Hier kommt der Vertrauensvorschuss der Rekursion (*Recursive Leap of Faith*) zum Tragen

Lösung zu "Power Set" (Konzeptionell)

Gegeben: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset
```

[2]

$$\mathcal{P}(\{b, c, d\}) = \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

[2] Hier kommt der Vertrauensvorschuss der Rekursion (*Recursive Leap of Faith*) zum Tragen

Lösung zu "Power Set" (Konzeptionell)

Gegeben: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset[2]
```

$$\mathcal{P}(\{b, c, d\}) = \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// init result with power set of remaining subset
```

[2] Hier kommt der Vertrauensvorschuss der Rekursion (*Recursive Leap of Faith*) zum Tragen

Lösung zu "Power Set" (Konzeptionell)

Gegeben: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset
```
[2]

$$\mathcal{P}(\{b, c, d\}) = \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// init result with power set of remaining subset
```

$$\texttt{result} \leftarrow \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

[2] Hier kommt der Vertrauensvorschuss der Rekursion (*Recursive Leap of Faith*) zum Tragen

Lösung zu "Power Set" (Konzeptionell)

Gegeben: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset[2]
```

$$\mathcal{P}(\{b, c, d\}) = \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// init result with power set of remaining subset
```

$$\texttt{result} \leftarrow \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// add first element to every set in the powerset
```

[2] Hier kommt der Vertrauensvorschuss der Rekursion (*Recursive Leap of Faith*) zum Tragen

Lösung zu "Power Set" (Konzeptionell)

Gegeben: $\{a, b, c, d\}$

```
// set has at least 1 element -> split set into two sets
```

$$\{a\}, \qquad \{b, c, d\}$$

```
// get power set for remaining subset[2]
```

$$\mathcal{P}(\{b, c, d\}) = \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// init result with power set of remaining subset
```

$$\texttt{result} \leftarrow \{\{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots\}$$

```
// add first element to every set in the powerset
```

$$\left\{ \begin{matrix} \{\}, \{b\}, \{c\}, \{d\}, \{b, c\}, \dots, \\ \{a\}, \{a, b\}, \{a, c\}, \{a, d\}, \{a, b, c\}, \dots \end{matrix} \right\}$$

[2] Hier kommt der Vertrauensvorschuss der Rekursion (*Recursive Leap of Faith*) zum Tragen

Lösung zu "Power Set" (Basisfall)

Lösung zu "Power Set" (Basisfall)

```
SetOfCharSets power_set(const CharSet& set) {
  // base case: empty set
  if (set.size() == 0) {
    return SetOfCharSets(CharSet());
  }
```

Lösung zu "Power Set"

```
// set has at least 1 element -> split set into two sets.
CharSet first_element_subset = CharSet(set.at(0));
CharSet remaining_subset = set - first_element_subset;

// get power set for remaining subset
SetOfCharSets remaining_subset_power_set = power_set(remaining_subset);

// init result with power set of remaining subset
SetOfCharSets result = remaining_subset_power_set;

// add first element to every set in the powerset
for (unsigned int i = 0; i < remaining_subset_power_set.size(); ++i) {
  result.insert(first_element_subset + remaining_subset_power_set.at(i));
}

return result;
```

Fragen/Unklarheiten?

7. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schöne Woche noch!