

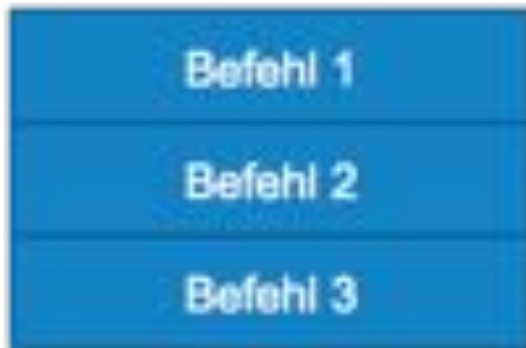
Algorithmen in der Sekundarstufe I



phsz

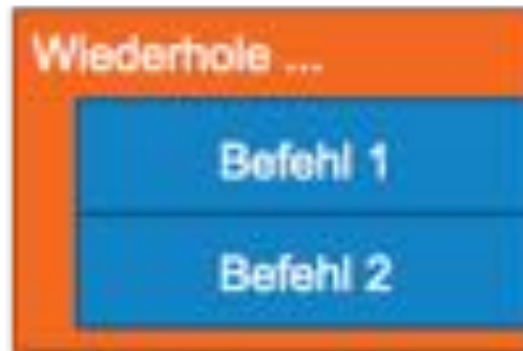
Am Ende der **Primarstufe** sollte Schüler/innen diese Grundkonzepte kennen und anwenden können:

Sequenz

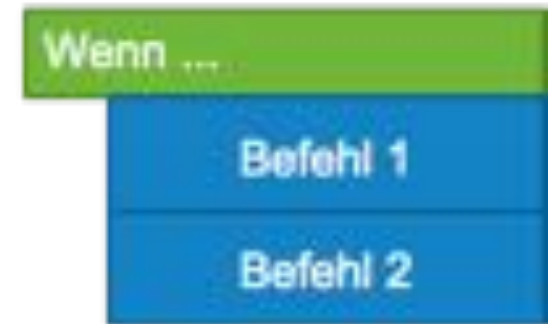


Abläufe planen und ausführen

Schleife



Bedingung



wenn Wand berührt, dann
Spiel verloren

Vom “Problem“ zum Programm in der **Sekundarstufe**

3

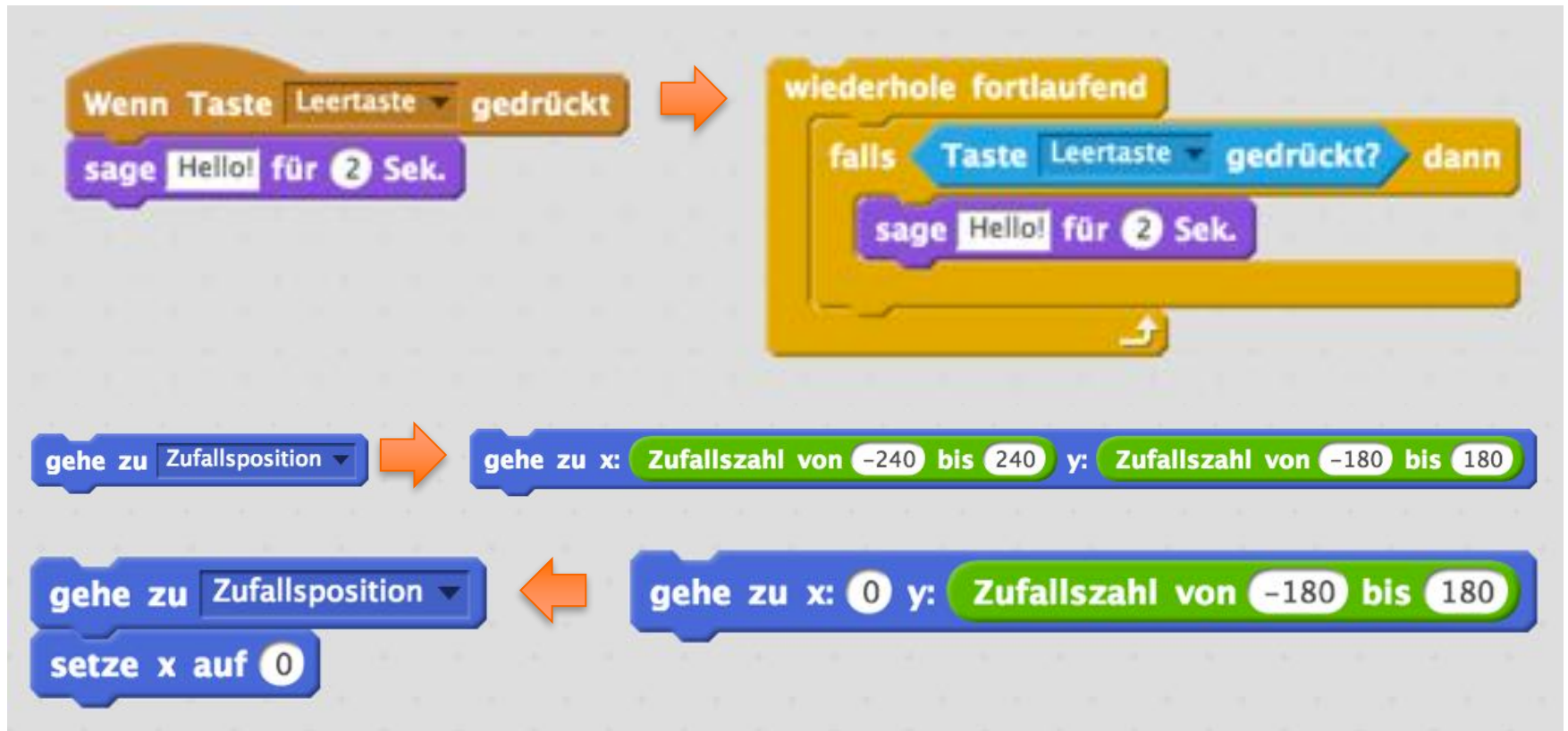
- g » können selbstentdeckte Lösungswege für einfache Probleme in Form von lauffähigen und korrekten Computerprogrammen mit Schleifen, bedingten Anweisungen und Parametern formulieren.
- h » können selbstentwickelte Algorithmen in Form von lauffähigen und korrekten Computerprogrammen mit Variablen und Unterprogrammen formulieren.
- i » können logische Operatoren verwenden (und, oder, nicht).

Probleme können sehr vielfältig sein:

- Wie berechne ich mit einem Programm die Fläche eines Dreiecks?
- Wie kann ich eine Figur am Bildschirm mit der Maus steuern?
- ...
 - eigenen Weg wählen, probieren und entdecken → Scratch

Abstraktionsstufen - Level of Abstraction

Scratch bietet bereits einige Blöcke an, die theoretisch aus anderen Blöcken bestehen könnten. Diese Blöcke erleichtern den Einstieg und können später durch „mächtigere“ Konstrukte ersetzt werden:



Variable wird implizit verwendet, aber nicht selbst definiert:



Die Variable „Antwort“ wird nicht selbst gesetzt, sondern automatisch über den „frage“-Block. Die „Antwort“ ist ein Block, das Prinzip der Variable wird aber nur gestreift.

Variable wird selbst definiert und als einfacher Zähler verwendet:



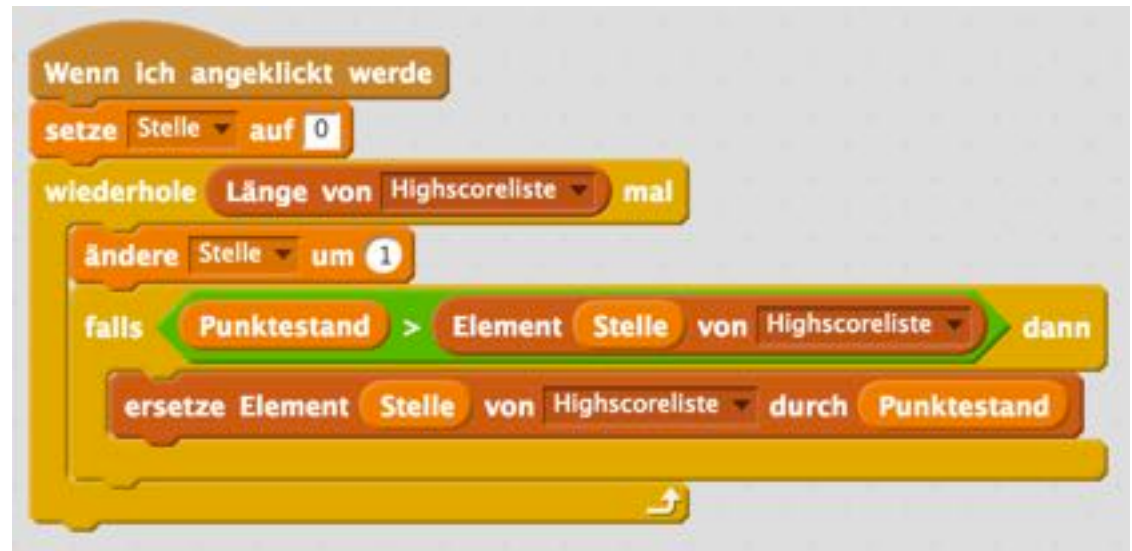
Die Variable „Punktestand“ wird selbst gesetzt und verändert. Sie wird jedoch noch nicht als Platzhalter in andere Blöcke eingesetzt.

Variable wird selbst definiert und als Platzhalter in andere Blöcke eingesetzt:



Hierfür braucht man ein Verständnis von der Variable als „Schachtel“, die zu unterschiedlichen Zeitpunkten einen unterschiedlichen Wert haben kann.

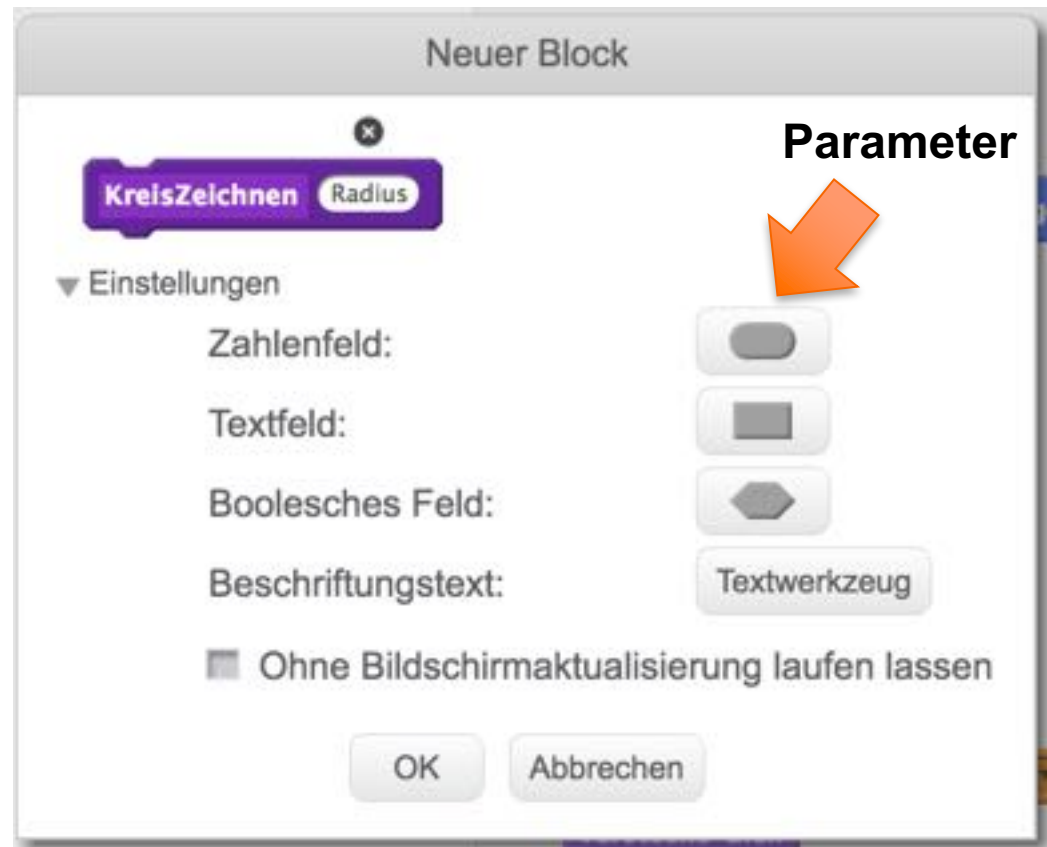
Listen = Sammlung mehrerer Variablen



Variable als Laufindex zum Suchen von Einträgen in der Liste. Zum Beispiel zum Ersetzen eines Eintrags an einer Stelle in der Liste.

- Prozeduren, Funktionen, **Unterprogramme** sind ein **Mittel der Abstraktion**:
 - *Wiederverwendung* – Es gibt im Projekt mehrere Stellen, wo die gleichen Befehle genutzt werden. An einer Stelle zusammenfassen = nur einmal korrigieren/ändern müssen.
 - *Schnittstelle* – Implementation unabhängig vom Rest möglich, solange Schnittstelle eingehalten wird (Blackbox-Prinzip).
 - *Parametrisierung* – ein Unterprogramm definiert mit Hilfe von Parametern eine ganze Klasse von Problemen und nicht nur einen spezifischen Fall.

Unterprogramm = eigener Block in Scratch



Eigene Blöcke - Beispiel



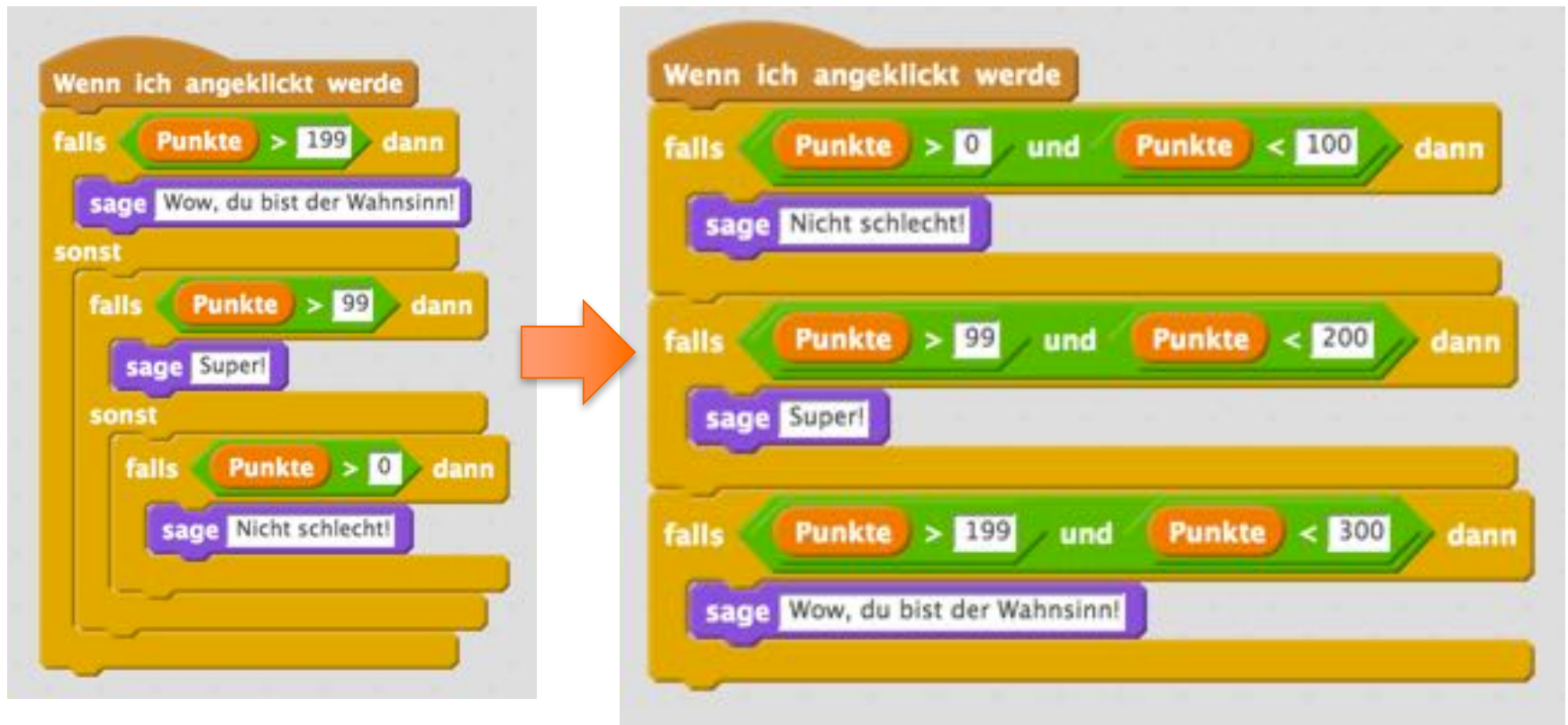
lokale Variable „Radius“

$$U = 2 \cdot \pi \cdot r$$



Logische Operatoren UND, ODER, NICHT verwenden

Logische Operatoren verbinden Wahrheitswerte.



Logische Operatoren UND, ODER, NICHT verwenden

falls (Leertaste gedrückt UND (NICHT (wird Farbe ... berührt))), dann:



(NICHT (... ODER ...))

Logische Operatoren UND, ODER, NICHT verwenden

Bei sehr verschachtelten Anweisungen wird die Erstellung des Programms fehleranfällig und schwer zu lesen

Beispiel ist keine Zielstellung für die Sekundarstufe I



The image shows a Scratch script for a game loop. It starts with a 'Wenn ich UPDATE PLAYER empfangen' block. Inside, there is a 'falls die = false und Lives > 0 dann' block. This block contains two nested 'falls' blocks. The first nested 'falls' block is 'falls nicht Friction = 0 und nicht Taste Pfeil nach oben gedrückt? dann', which contains 'setze VelX auf VelX * Friction' and 'setze VelY auf VelY * Friction'. The second nested 'falls' block is 'falls Taste Pfeil nach links gedrückt? dann', which contains 'ändere VelR um Turn Speed * -1', 'falls -9 > VelR dann', and 'setze VelR auf -9'. In the bottom right corner, there is a small screenshot of a game window titled 'Apollos - Extrembeispiel von PHSZ (unveröffentlicht)'. The game screen shows a dark space with green stars and the text 'APOLLOS PRESS Z TO START'.

Aufgabe zum Thema Unterprogramm und Variablen

Scratch - Arbeitsblätter

Hütchenspieler

1

Zahlenmauer

Computer rechnen sehr schnell
Eine große Stärke des Computers ist seine Fähigkeit sehr schnell zu rechnen. Vielleicht hast du Schule bereits Zahlenmauern kennengelernt. Jeder Stein der Mauer steht für eine Zahl. Die Zahl jedes Steins wird aus der Summe der beiden darunterliegenden Steine gebildet. In der einfachsten Variante ist die unterste Reihe von zwei Steinen. In diesem Experiment vorgegeben und die restlichen Zahlen sollen ausgerechnet werden. In diesem Experiment wollen wir ausprobieren, wie schnell der Computer eine Zahlenmauer berechnen kann.

Aus Sicht der Mathematik können wir eine Zahlenmauer als Folge von einfachen Gleichungen beschreiben, unabhängig davon, welche Zahlen konkret darin stehen (Stein₂ ist z.B. der erste Stein in der zweiten Reihe):
Stein₂ = Stein₁ + Stein₁ Stein₃ = Stein₂ + Stein₂
Stein₃ = Stein₁ + Stein₁ Stein₄ = Stein₂ + Stein₂
Stein₄ = Stein₁ + Stein₁ Stein₄ = Stein₃ + Stein₃

Eine Welt aus Variablen
In Scratch definieren wir für jeden Stein eine Variable. Wechseln wir die Blockpalette auf „Daten“ und klicke auf **Neue Variable**, um eine neue Variable zu erstellen. Erstelle insgesamt 10 Variablen mit passenden Namen für die einzelnen Steine. Diese erscheinen automatisch oben links auf der Bühne. Einzelnen Steine. Diese erscheinen automatisch oben links auf der Bühne. Einzelnen Steine. Diese erscheinen automatisch oben links auf der Bühne. Einzelnen Steine. Diese erscheinen automatisch oben links auf der Bühne.

Erstelle eine neue Figur **Hüte**, welche beim Anklicken eine neue zufällige Zahlenmauer berechnet. Erstelle dazu ein Skript, welches die Variablen der unteren 4 Steine der Mauer mit einer Zufallszahl befüllt und alle weiteren Steine aus der Summe der beiden vorherigen berechnet (siehe Skript links). Wenn du alles richtig gemacht hast, sollte eine fertig gebaute Zahlenmauer entstehen. Sollte etwas nicht stimmen, prüfe nochmals das Skript und die Anordnung der Variablen auf der Bühne.


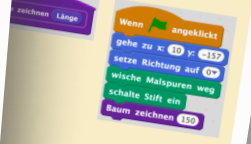
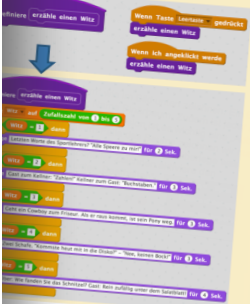
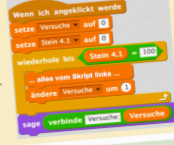

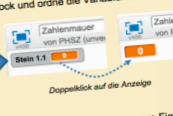

Das geht aber schnell
Für Aufgabenblätter wäre es schön Zahlenmauern mit einer bestimmten Zahl im Deckstein zu erzeugen. Erstelle eine weiteren Schaltfläche, welche Zahlenmauern mit 100 erzeugt. Zähle die Versuche, die der Computer unternommen hat, bis zufällig eine passende Mauer entstanden ist.

Zusatz
Verändere das zweite Skript so, dass Zahlenmauern mit 100'000 im Deckstein entstehen. Welche Anpassungen musst du zusätzlich vornehmen, damit eine Zahlenmauer mit 100'000 entstehen kann? Kannst du die Berechnungszeit messen? Verwende **Zeige Steuerelement** und **Warte**.

2

3

4



Probleme anders lösen ...

Zahlenmauern mit Deckstein 100'000 zufällig generieren statt selber zu rechnen.

Nicht immer wäre eine Berechnungsvorschrift so einfach findbar.

„Blockchains“ (z.B. Bitcoins) basieren sogar auf das zufällige Würfeln von kryptographischen Zahlen und belohnen den „Miner“ im Erfolgsfall.

Versuche:300260 in
4.204Sekunden

Versuche:29459 in
0.4Sekunden

Versuche:72756 in
1.116Sekunden

100'000

Scratch - Arbeitsblätter
Zahlenmauer phsz 1

Computer rechnen sehr schnell
Eine grosse Stärke des Computers ist seine Fähigkeit sehr sehr schnell zu rechnen. Vielleicht hast du Schule bereits Zahlenmauern kennengelernt. Jeder Stein der Mauer steht für eine Zahl. Die Zahl jedes Steins wird aus der Summe der beiden darunterliegenden Steine gebildet. In der einfachsten Variante ist die unterste Reihe vorgegeben und die restlichen Zahlen sollen ausgerechnet werden. In diesem Experiment wollen wir ausprobieren, wie schnell der Computer eine Zahlenmauer berechnen kann.

Aus Sicht der Mathematik können wir eine Zahlenmauer als Folge von einfachen Gleichungen beschreiben, unabhängig davon, welche Zahlen konkret darin stehen (Stein₂ ist z.B. der erste Stein in der zweiten Reihe):
Stein₂ = Stein₁ + Stein₁ Stein₃ = Stein₂ + Stein₂
Stein₂ = Stein₁ + Stein₁ Stein₃ = Stein₂ + Stein₂
Stein₂ = Stein₁ + Stein₁ Stein₄ = Stein₃ + Stein₃

Eine Welt aus Variablen
In Scratch definieren wir für jeden Stein eine Variable. Wechsel in der Blockpalette auf „Daten“ und klicke auf „Neue Variable“, um eine neue Variable zu erstellen. Erstelle insgesamt 10 Variablen mit passenden Namen für die einzelnen Steine. Diese erscheinen automatisch oben links auf der Bühne. Verändere die Darstellung zu einem Block und ordne die Variablen auf der Bühne per Drag&Drop als Mauer an.

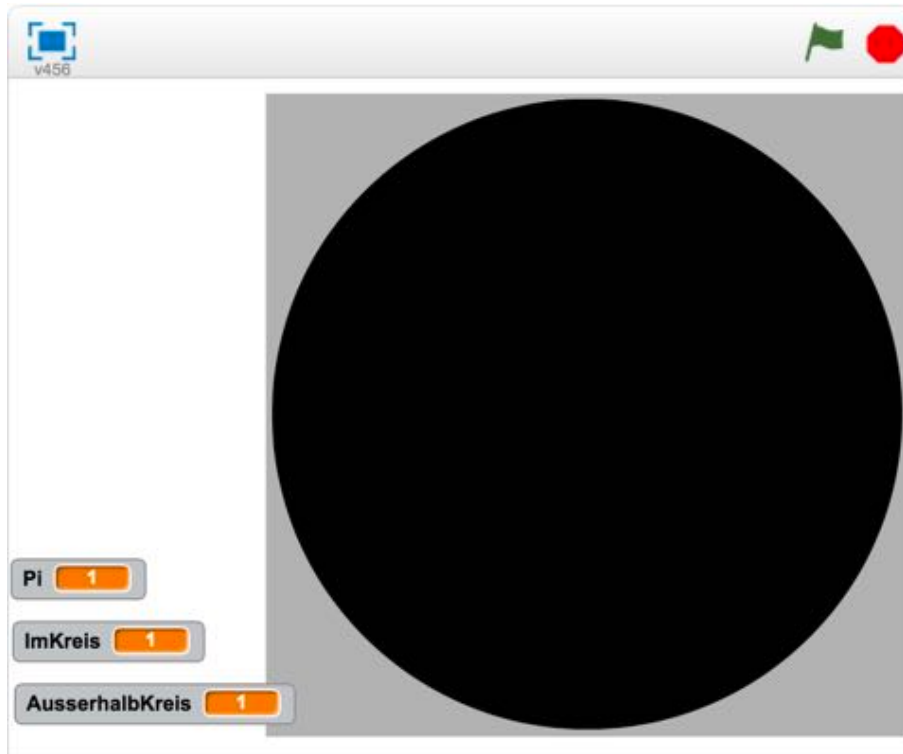
Erstelle eine neue Figur „Neu“, welche beim Anklicken eine neue zufällige Zahlenmauer berechnet. Erstelle dazu ein Skript, welches die Variablen der unteren 4 Steine der Mauer mit einer Zufallszahl befüllt und alle weiteren Steine aus der Summe der beiden vorherigen berechnet (siehe Skript links). Wenn du alles richtig gemacht hast, sollte eine fertig gelöste Zahlenmauer entstehen. Sollte etwas nicht stimmen, prüfe nochmals das Skript und die Anordnung der Variablen auf der Bühne.

Das geht aber schnell
Für Aufgabenblätter wäre es schön Zahlenmauern mit einer bestimmten Zahl im Deckstein zu erzeugen. Erstelle eine weiteren Schaltfläche, welche Zahlenmauern mit 100 erzeugt. Zähle die Versuche, die der Computer unternommen hat, bis zufällig eine passende Mauer entstanden ist.

Zusatz
Verändere das zweite Skript so, dass Zahlenmauern mit 100'000 im Deckstein entstehen. Teste das Programm. Welche Anpassungen musst du zusätzlich vornehmen, damit eine Zahlenmauer mit 100'000 entstehen kann? Kannst du die Berechnungszeit messen? Verwende `Zeit messen` und `Zeit`.

<https://scratch.mit.edu/projects/168141729>

Beispiel: Bestimmung von π mit Simulation



<https://scratch.mit.edu/projects/168273389>

Algorithmen vergleichen: Sortieren und Finden

- i » können verschiedene Algorithmen zur Lösung desselben Problems vergleichen und beurteilen (z.B. lineare und binäre Suche, Sortierverfahren).



Beispiel: Lineare Suche vs. Binär-Suche unplugged

Wörterbuch oder Lexikon (= vorsortierte Einträge) mitbringen.



Beispiel: Lineare Suche vs. Binär-Suche unplugged

Vorgehen – unplugged Beispiel mit einem Lexikon / Duden:

- SuS nennen einen zufälligen Begriff – zum Beispiel *Sonne*.
- LP fängt auf der ersten Seite im Buch an und fährt mit dem Finger über die Seite (zeilenweise, jeden Eintrag abfahren). Etwas Ausdauer zeigen 😊
- SuS sollen merken, dass dieses Suchverfahren ineffizient ist und viel zu lange dauert.
- Vorschläge sammeln, wie man das gesuchte Wort schneller finden könnte. → Sortierung hilft – wir können bereits zum Anfangsbuchstaben S springen.
- Bei Anfangsbuchstaben S erneut beginnen mit dem Finger zeilenweise abzufahren. Erneut ineffizient, der zweite Buchstabe ist aber nicht direkt über ein Register ablesbar im Buch
- → die Kinder sollen auf die Idee kommen, in die Mitte von „S“ zu blättern und dann zu schauen ob sie vorher oder nachher weitersuchen müssen.

Beispiel – lineare Suche

Bei der **linearen Suche** beginnt man am Anfang der Liste und schaut jedes Element an, bis man das gesuchte gefunden hat.

Beispiel Such nach dem Eintrag „O“



10 Einträge wurden angeschaut und verglichen

Eine Liste mit 10'000 Einträgen:

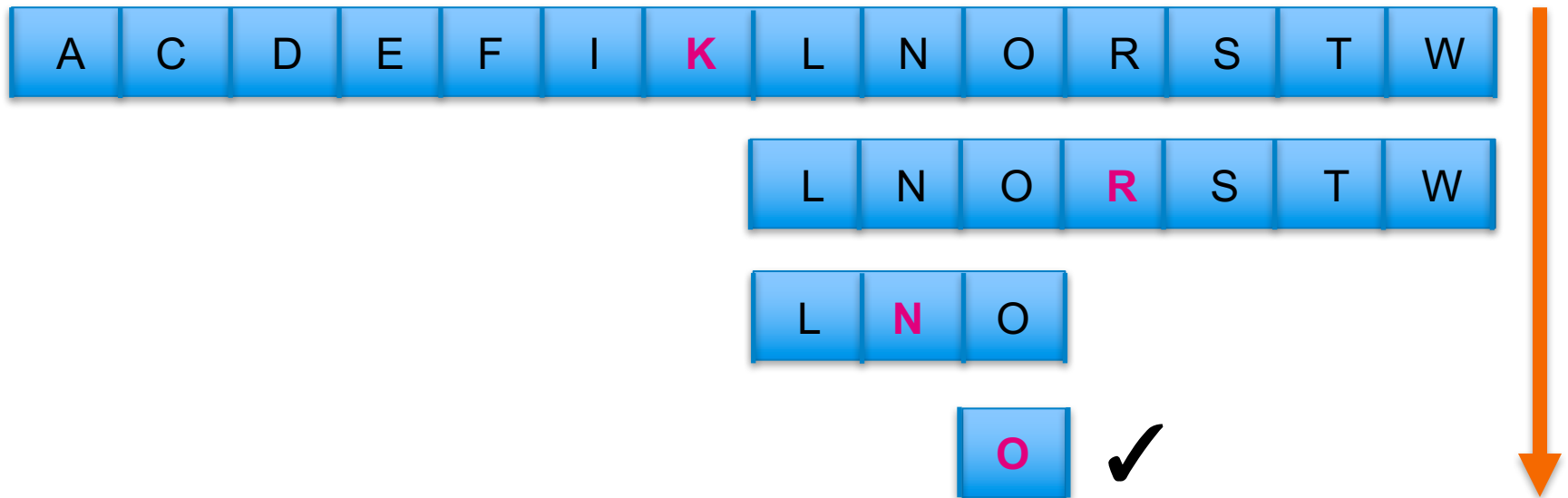
- Wie viele Einträge müssten wir im schlimmsten Fall anschauen?
- Wie viele Einträge müssten wir im besten Fall anschauen?
- Wie viele Einträge müssten wir durchschnittlich anschauen?

Beispiel – Binärsuche

Bei der **Binärsuche** teilt man die vorsortierte Liste immer in zwei Hälften.

→ binäre Entscheidung, ob man mit der einen oder anderen Hälfte vorsetzt.

Beispiel Such nach dem Eintrag „O“



4 Einträge wurden angeschaut und verglichen

Eine Liste mit 10'000 Einträgen? schlimmsten Fall, bester Fall, durchschnittlich?

Beispiel – lineare Suche vs. Binärsuche

Erkenntnisse:

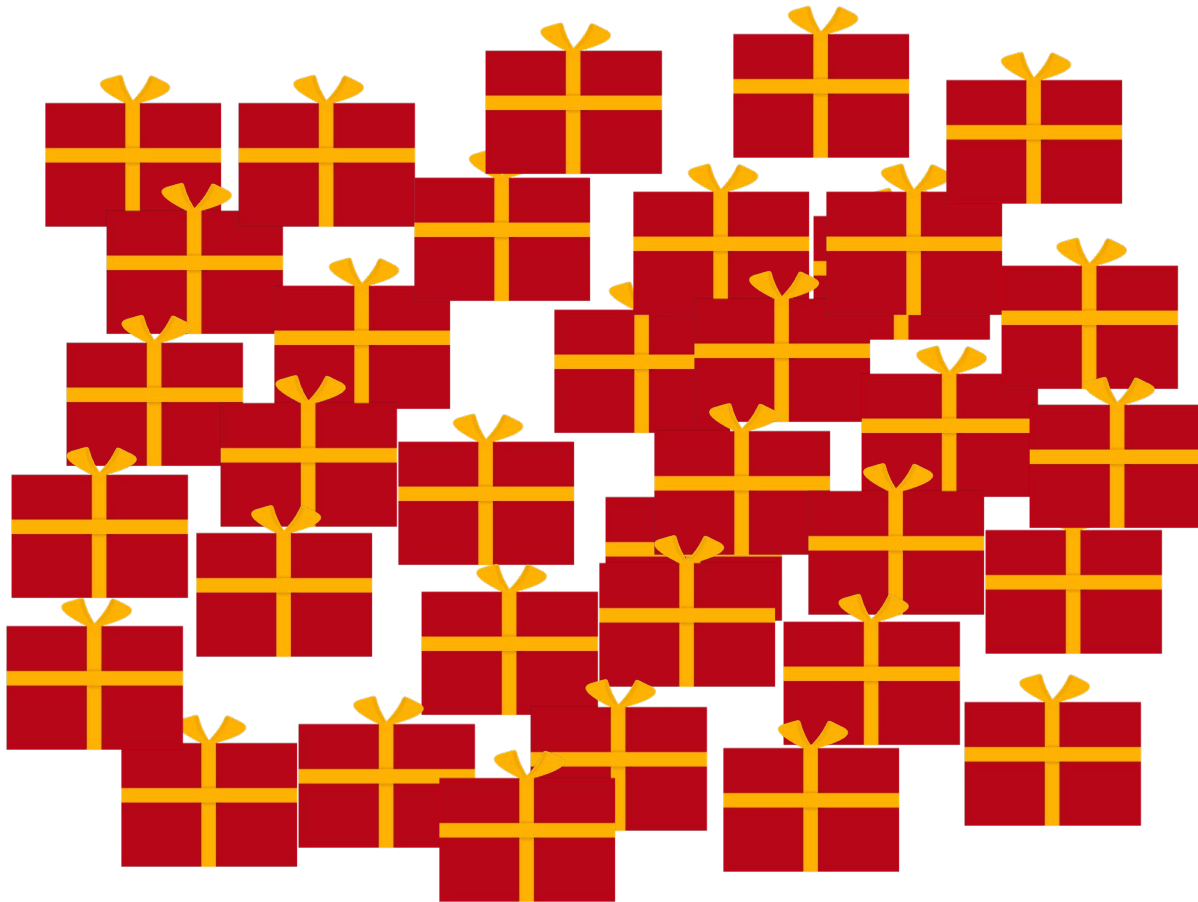
- Obwohl beide Algorithmen genau das gleiche Ergebnis (den Eintrag „O“) liefern, sind sie unterschiedlich schnell → effizient
- Als Nutzer sehen wir einem Programm in der Regel nicht an, welche Algorithmen eingesetzt werden, wir sehen nur das Ergebnis „O“.

Algorithmen können auch in anderen
Situationen helfen → Beispiel



Binärsuche - Anwendungsbeispiel

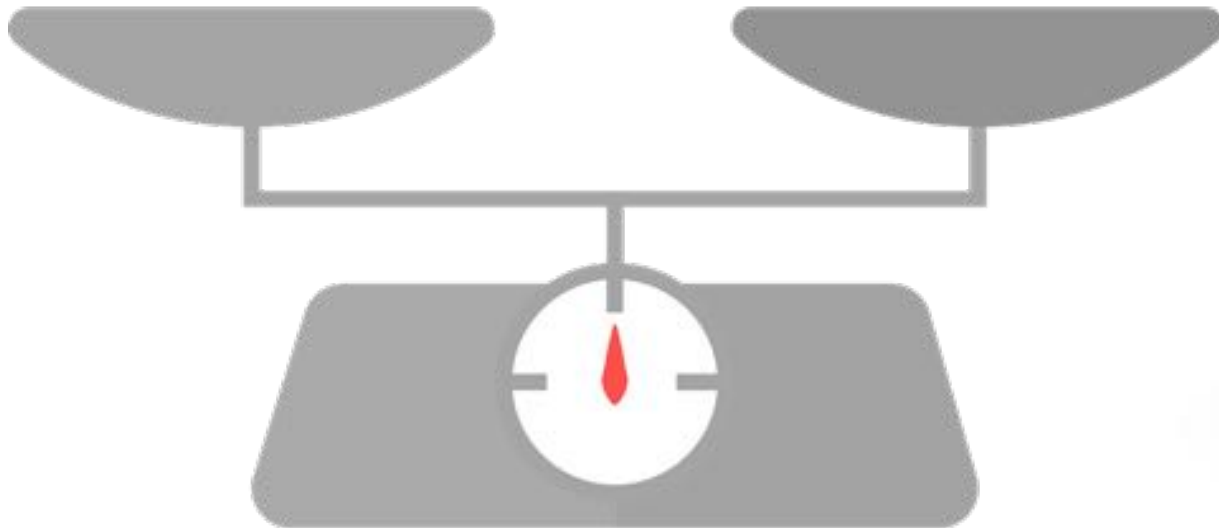
1000x  mit dem gleichen Spielzeugauto  darin

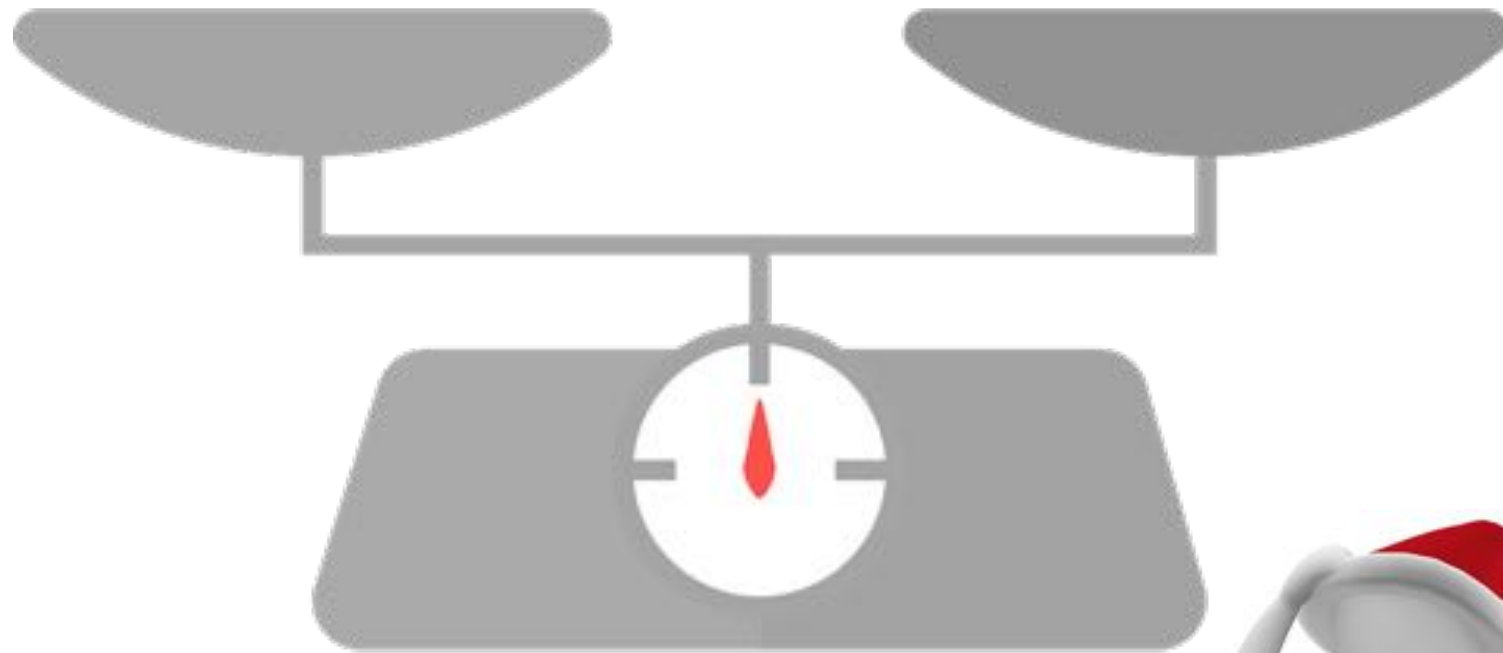


Oh, wo kommt denn der her?
Der fehlt jetzt in einem Geschenk!
Aber in welchem ???

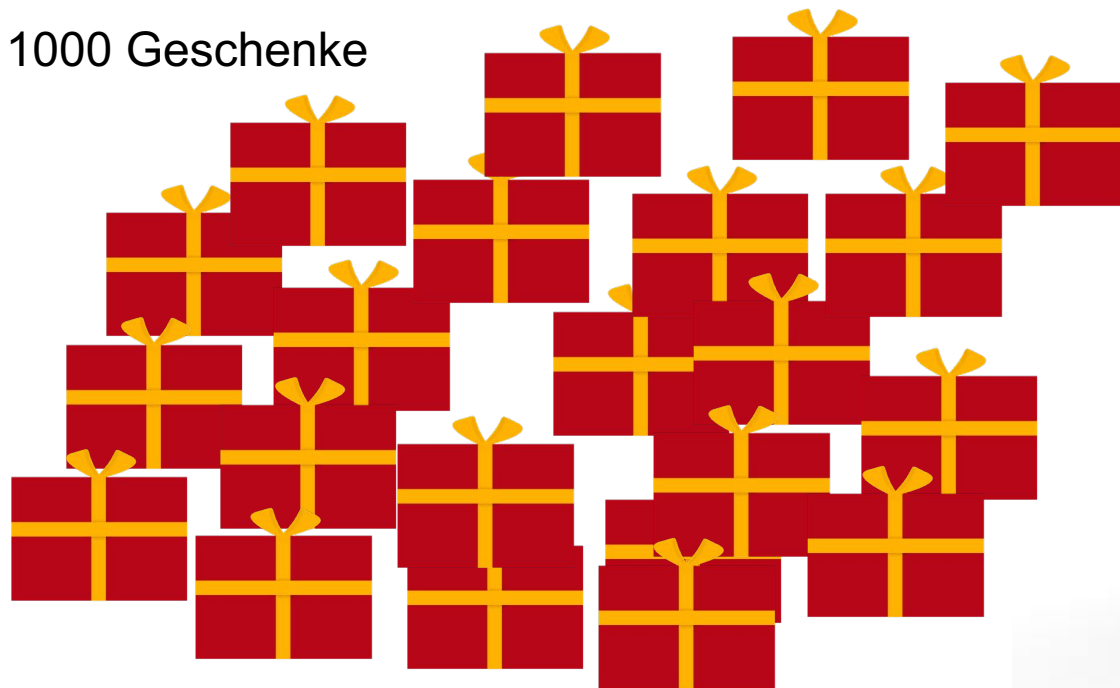


Kein Problem, wir müssen
die Geschenke doch nur **10 Mal** mit
unserer riesigen Waage abwiegen,
dann wissen wir in welchem
der Schlüssel fehlt !



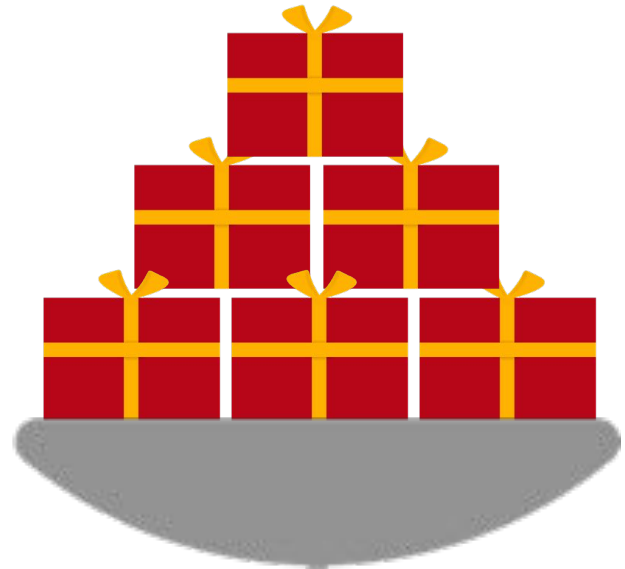
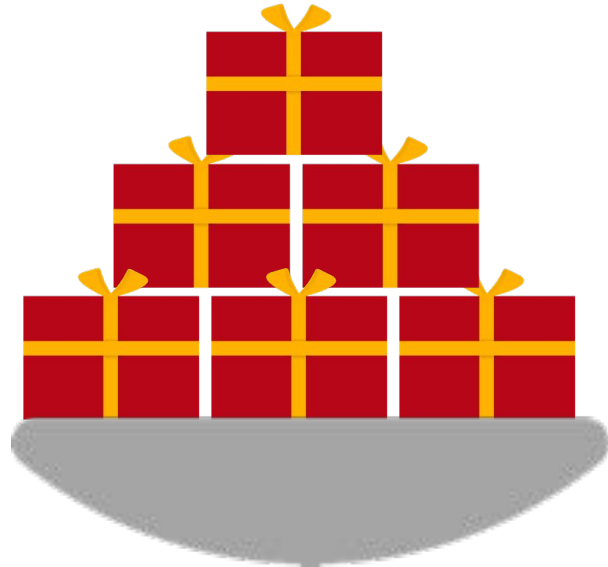


1000 Geschenke



500 Geschenke

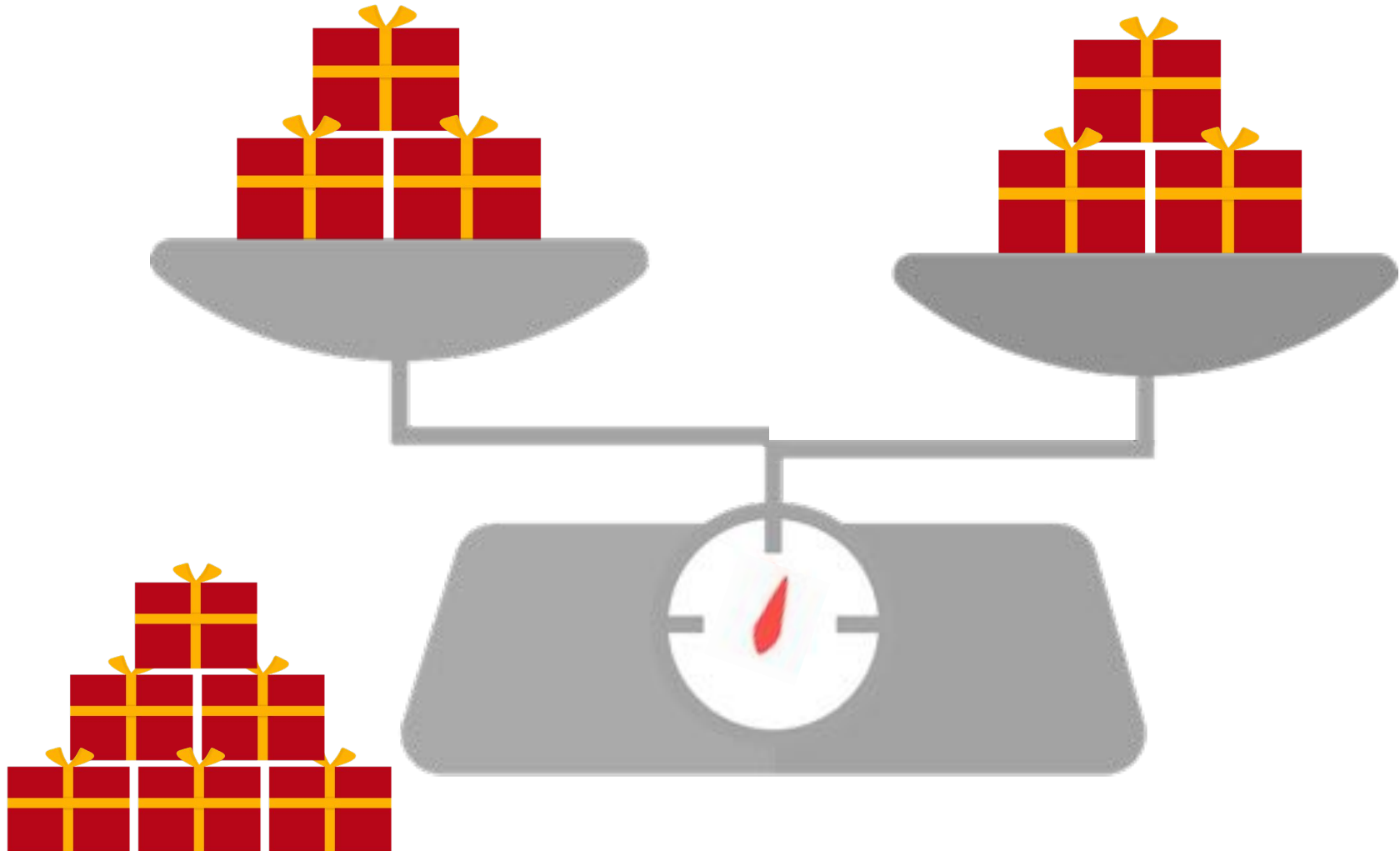
500 Geschenke



leichter = hier fehlt etwas

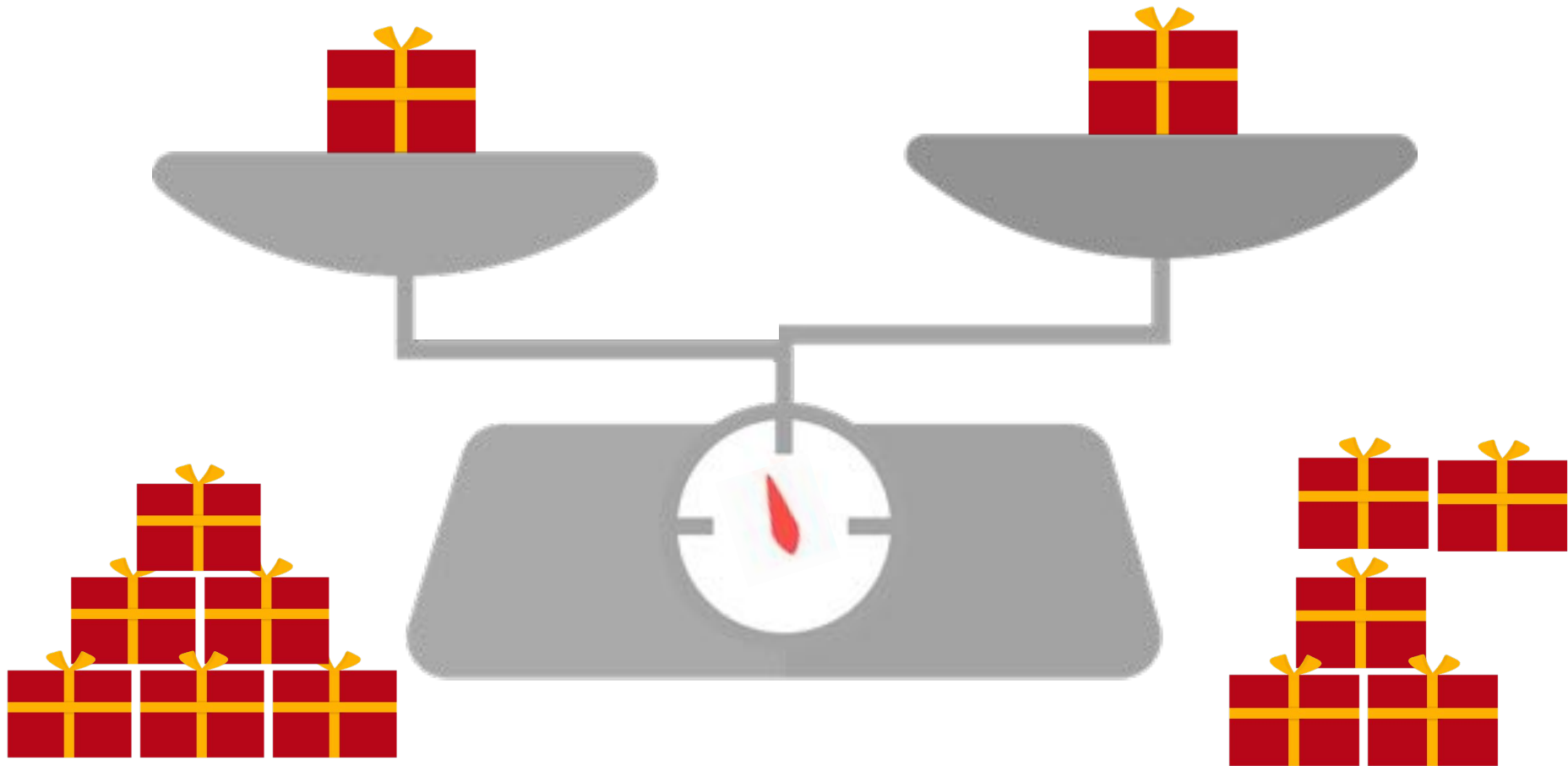
250 Geschenke

250 Geschenke



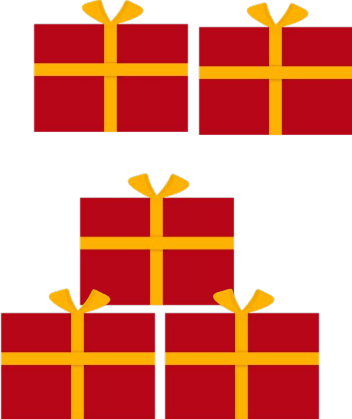
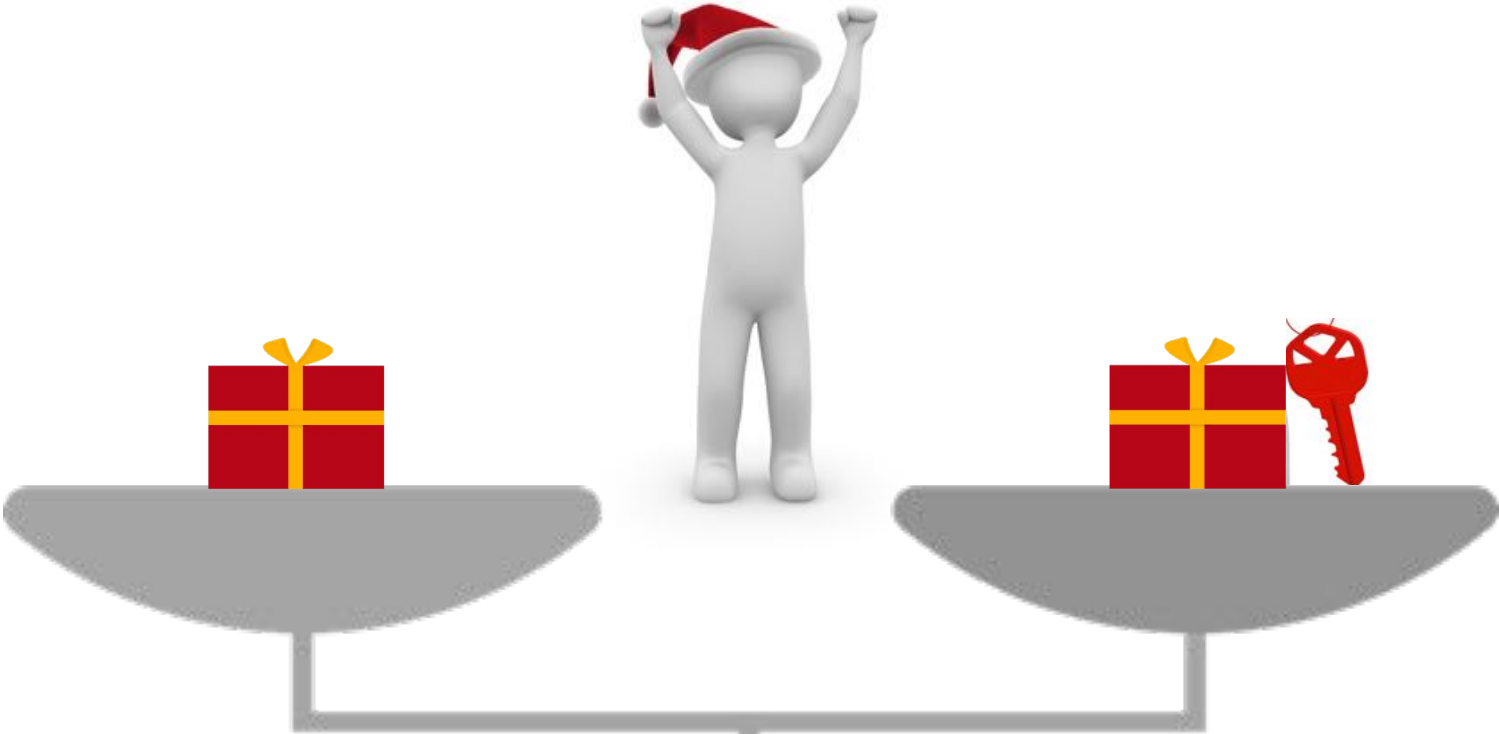
1 Geschenk

1 Geschenk



1 Geschenk

1 Geschenk





Algorithmen Vergleichen - Beispiel Sortieren und Suchen

- i » können verschiedene Algorithmen zur Lösung desselben Problems verglichen und beurteilt (z.B. lineare und binäre Suche, Sortierverfahren).

INSERTION-Sort (kurzes Scratch-Programm)

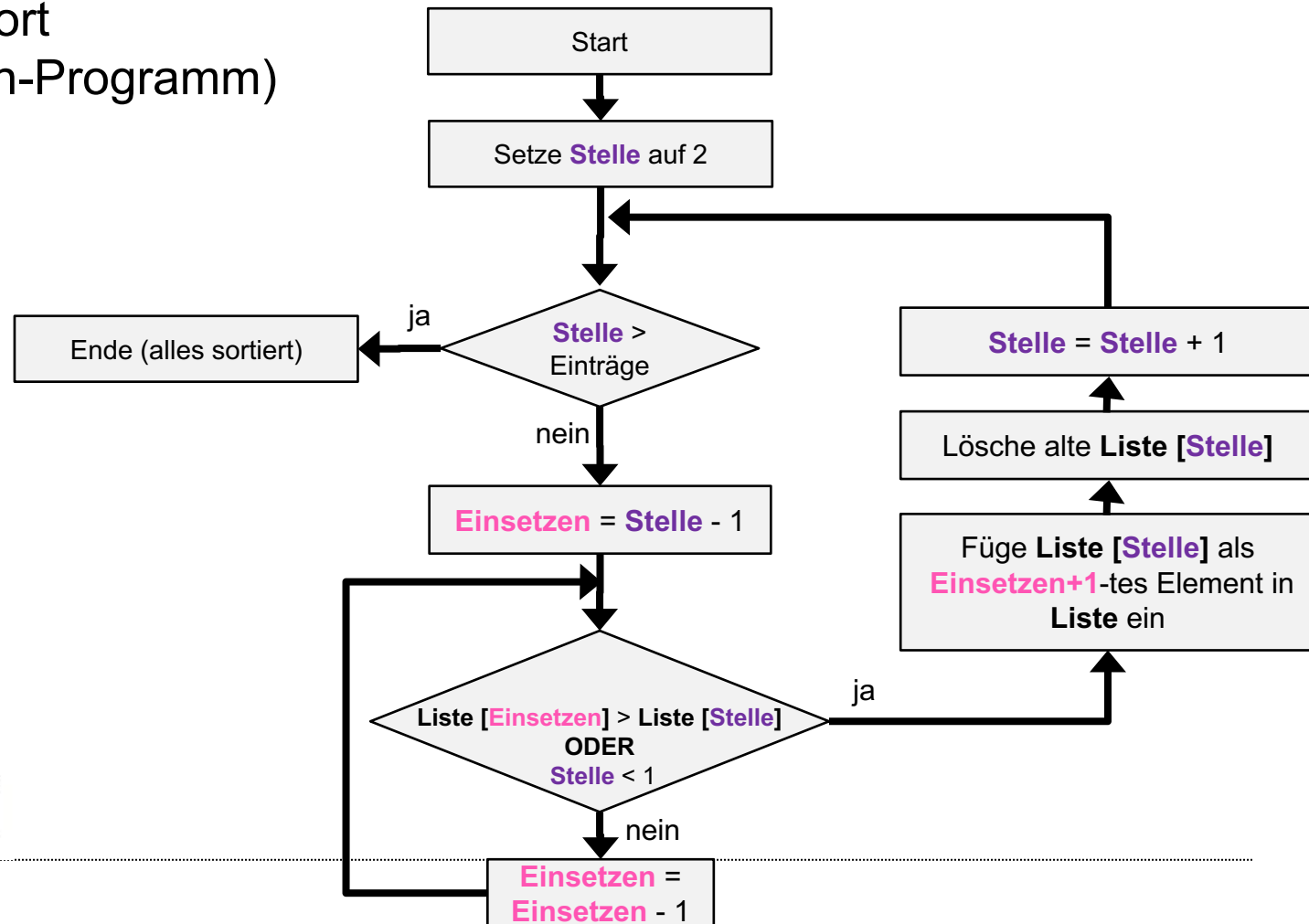
Liste

1	26
2	59
3	39
4	96
5	79
6	78
7	7
8	40

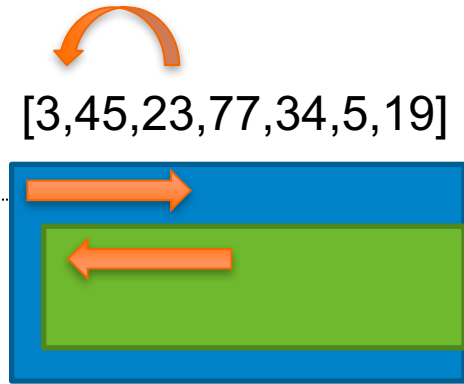
+ Länge: 20

Stelle

Einsetzen



Sortieren in Scratch – INSERTION Sort

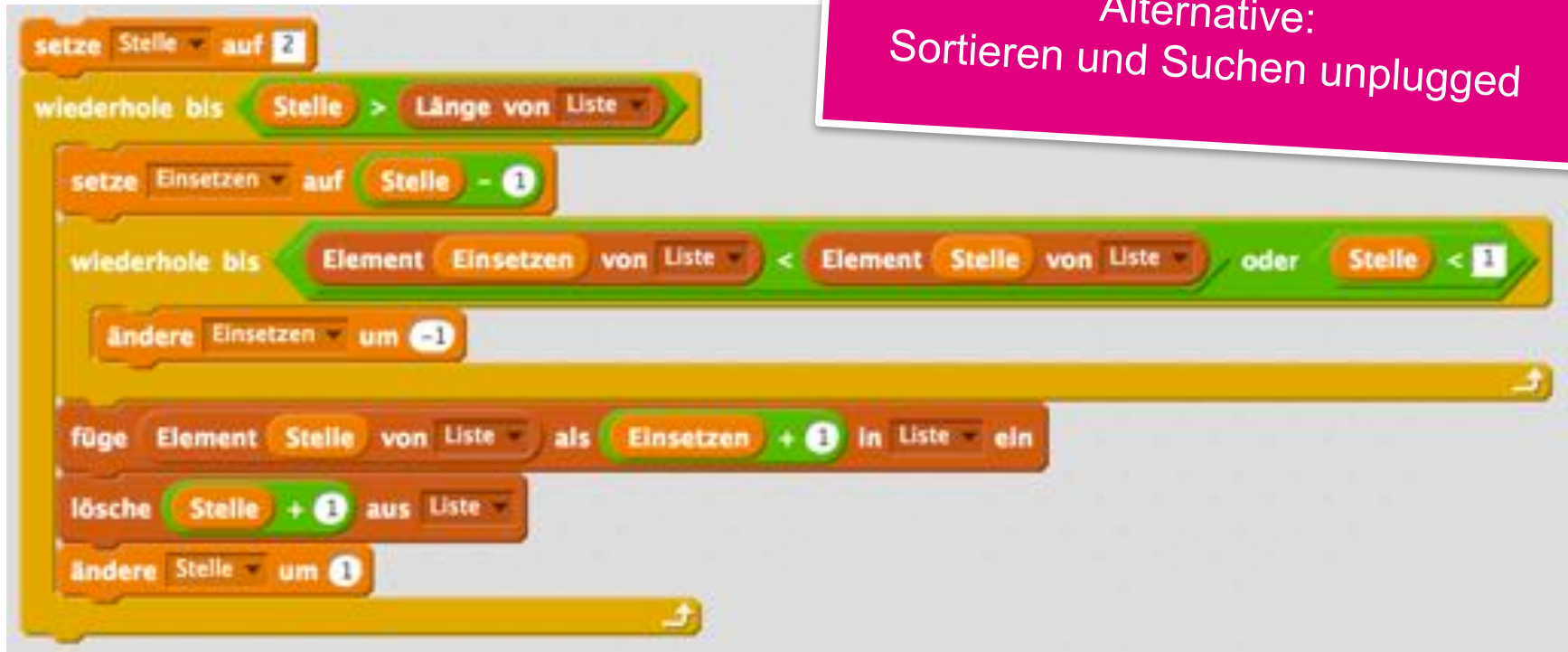


In Worten:

- Schritt 1:
Wir vergleichen den Wert an der aktuelle Stelle mit allen vorherigen, bis wir einen kleineren Wert gefunden haben, oder am Anfang angekommen sind.
- Schritt 2:
Wurde ein kleineren Wert gefunden, wir das aktuelle Element in die Liste NACH dieser Stelle eingefügt. Anschliessend wird das aktuelle Element aus der Liste gelöscht (wir haben es ja an die andere Stelle „kopiert“).
- Schritt 3:
die aktuelle Stelle wird um +1 erhöht und mit Schritt 1 fortgesetzt, solange bis wir am Ende der Liste angekommen sind.

Sortieren in Scratch – INSERTION Sort

Alternative:
Sortieren und Suchen unplugged



Dieses Verfahren umzusetzen ist eventuell nicht zielstufengerecht. Man könnte es als vorgegebenen Block oder Scratch-Programm bereitstellen. Scratch ist mit seiner Darstellung wird mit wachsender Komplexität schwieriger zu lesen und zu verstehen.

Sortieren und Finden – unplugged Verkehrshaus (iFactory)

- i » können verschiedene Algorithmen zur Lösung desselben Problems vergleichen und beurteilen (z.B. lineare und binäre Suche, Sortierverfahren).

