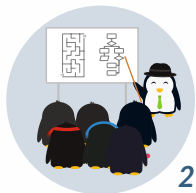


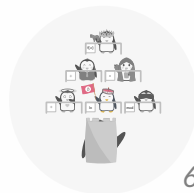
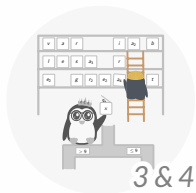
I've built this. With my *bare* hands!

Tutorium Eins

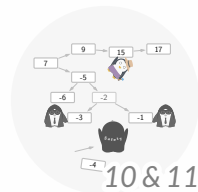
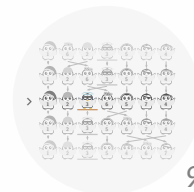
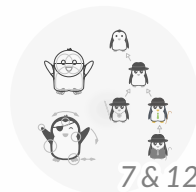
Theorie



Grundlagen



Vertiefungen



*Practice yourself, for heaven's sake in little things, and
then proceed to greater.
— Epictetus [EpiAD, adapted, chp. 16]*

Präsenzaufgabe

1

It's true, isn't it?

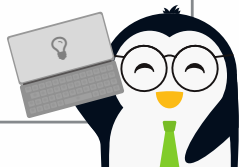
```
class BoolescheAusdruecke {  
    public static void main(String[] args) {  
  
    }  
}
```

Erweitern Sie die Datei.
Deklarieren und initialisieren
sie benötigte Variablen mit
„beliebigen“ Werten.

Konstruieren Sie boolesche Ausdrücke, die folgendes abprüfen:

- Die Zimmertemperatur (`temperatur`) beträgt höchstens 22,5 Grad Celsius.
- Eine Person (`alter`) ist nicht zwischen 13 und 18 Jahre alt.

Geben Sie die Ergebnisse über `System.out.println(<...>)` aus.



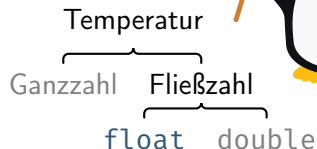
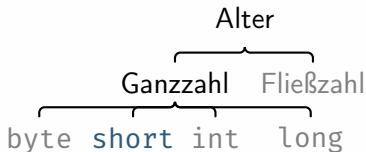
Die Datentypen finden

Zimmertemperatur höchstens 22,5 Grad Celsius, Alter nicht zwischen 13 und 18 Jahren.

```
class BoolescheAusdruecke {  
    public static void main(String[] args) {  
        float temperatur = 18.0f;  
        short alter = 15;  
        // ...  
    }  
}
```

Einen generell korrekten Datentyp gibt es selten. Diverse Bedingungen hängen vom Kontext ab.

Die Zahlen sind erstmal beliebig.



Boolesche Ausdrücke konstruieren

Zimmertemperatur höchstens 22,5 Grad Celsius, Alter nicht zwischen 13 und 18 Jahren.

```
class BoolescheAusdruecke {  
    public static void main(String[] args) {  
        float temperatur = 18.0f;  
        short alter = 15;  
        System.out.println(temperatur <= 22.5f);  
        System.out.println(!(13 <= alter && alter <= 18));  
    }  
}
```

BoolescheAusdruecke.java

Deklarieren, Initialisieren und Zuweisen

- › Die Begriffe sind leicht definiert:

Deklaration: Reserviert einen Namen und setzt den Typ

`int x, y;`

Initialisierung: Die *erste* Zuweisung einer Variable

(Erst jetzt ist die Variable verwendbar)

Zuweisung: Setzen/Ändern des Wertes einer Variable

`x = 3;`

```
public static void main(String[] args) {
```

```
    int apfelsine;
```

Deklaration einer Variable „apfelsine“ vom Typ int

```
    char zeichen = 'x';
```

*Deklaration mit Typ char und **Initialisierung** mit dem Wert 'x'*

```
    apfelsine = 42;
```

Initialisierung mit dem Wert 42

```
    if(apfelsine % 2 == 0)
```

```
        zeichen = 'k';
```

Zuweisung des Wertes 'k'

```
}
```

Übungsblatt 1

Aufgabe 1: Aufwandsanalyse

In der Vorlesung haben wir uns mit der Aufwandsanalyse von Algorithmen beschäftigt und uns angeschaut, wie viele Operationen im schlechtesten Fall notwendig sind, um den Algorithmus durchzuführen. Im Folgenden wollen wir uns nun anschauen, wie sich solche Aussagen (vereinfacht) auf Maschinenoperationen übertragen lassen. Bei einer Firma fällt im Produktionsprozess täglich ein Optimierungsproblem an, für dessen Lösung eine Stunde zur Verfügung steht. Der Algorithmus zur Lösung des Optimierungsproblems muss bei Eingabe eines Problems der Größe n insgesamt $100n^2$ viele Operationen durchführen. Bisher verwendet die Firma zur Lösung einen Mikro-Rechner mit einem Prozessortakt von 800 MHz. Dieser kann in jeder Sekunde 40 000 der oben genannten Operationen berechnen.

- a) Welche Problemgröße n dürfen die Probleme maximal haben, damit die Berechnung innerhalb der Frist durchgeführt werden kann?
- b) Nun soll auf einen schnelleren Rechner mit 3 200 MHz Taktfrequenz umgestiegen werden. Nehmen Sie an, die höhere Taktfrequenz übersetze sich direkt in eine entsprechend höhere Rechengeschwindigkeit. Wie groß können nun die Optimierungsprobleme sein?
- c) Anstelle einer CPU mit einem, soll eine CPU mit 16 Kernen (jeweils mit 3 200 MHz) eingesetzt werden. Glücklicherweise lässt sich das Optimierungsproblem leicht parallelisieren, so dass sich die Geschwindigkeit im Vergleich zu einem Kern verneunfacht. Wie groß können nun die Optimierungsprobleme sein?

a) *Was ist die maximale Problemgröße n um in der Frist zu bleiben?*

In der Stunde haben wir $60 \cdot 60$ Sekunden und damit $60 \cdot 60 \cdot 40\,000$ Operationen. Bei Problemgröße n benötigen wir $100n^2$ Operationen:

$$100n^2 \leq 60 \cdot 60 \cdot 40\,000$$

$$100n^2 \leq 1\,440\,000 \quad \quad \quad | \cdot 1/100 \quad | \sqrt{}$$

$$n \leq \sqrt{1\,440\,000}$$

$$n \leq 1200$$

> Die Problemgröße n darf 1200 nicht übersteigen.

Die Macht der Taktfrequenz

- b) Angenommen, die höhere Taktfrequenz übersetze sich direkt in eine entsprechend höhere Rechengeschwindigkeit. Wie groß können die Optimierungsprobleme bei 3 200 MHz Taktfrequenz sein?

Bisher schaffen wir 40 000 Operationen, nun schaffen wir $40\,000 \cdot 3\,200\text{ MHz}/800\text{ MHz}$:

$$100n^2 \leq 60 \cdot 60 \cdot 40\,000 \cdot 3\,200\text{ MHz}/800\text{ MHz}$$

$$100n^2 \leq 576\,000\,000 \quad | \cdot 1/100 \quad | \sqrt{}$$

$$n \leq \sqrt{5\,760\,000}$$

$$n \leq 2400$$

> Die Problemgröße n darf nun 2400 nicht übersteigen.

Ein vervierfachen der Leistung verdoppelt die mögliche Problemgröße aufgrund des quadratischen Wachstums.

- c) Anstelle einer CPU mit einem, soll eine CPU mit 16 Kernen (jeweils mit 3 200 MHz) eingesetzt werden. Glücklicherweise lässt sich das Optimierungsproblem leicht parallelisieren, so dass sich die Geschwindigkeit im Vergleich zu einem Kern verneunfacht. Wie groß können nun die Optimierungsprobleme sein?

Die Anzahl der Operationen verneunfacht sich direkt:

$$100n^2 \leq 60 \cdot 60 \cdot 40\,000 \cdot 3\,200 \text{ MHz} / 800 \text{ MHz}$$

$$100n^2 \leq 5\,184\,000\,000 \quad | \cdot 1/100 \quad | \sqrt{}$$

$$n \leq \sqrt{51\,840\,000}$$

$$n \leq 7200$$

› Die Problemgröße n darf nun 7200 nicht übersteigen.

- Wir bekommen in der Regel (leider) nichts geschenkt.
 - Parallelisierung ist mit Kosten verbunden.
 - Wir müssen die parallelen Abläufe koordinieren, abgleichen, ...
 - Je nach Optimierung und Problem kann mehr Parallelisierung sogar schädlich sein!
 - Auch ist nie alles parallelisierbar (Initialisierungen, Ausgaben, ...).
Eine pessimistische Abschätzung liefert das „Amdahlsche Gesetz“ (wer mag: [Amd67]).
- Für die Skalierung hat n^2 einen viel größeren Einfluss, als der Faktor 100
 - So vervielfachen 16 CPUs mit je vierfacher Leistung die Performanz um lediglich 6!
 - Meist interessiert uns nur das Wachstumsverhalten (polynomiell, exponentiell, ...)

Aufgabe 2: Korrektheit von Algorithmen

Betrachten Sie die Anweisungsfolge. Handelt es sich hierbei um einen Algorithmus? Prüfen Sie alle notwendigen Voraussetzungen.

```
1: Setze      x = 1
2: solange    x ≥ 0 wiederhole
3:            Verdopple x
4:            Verringere x um 1
5: ende
```

Ausführ- und Reproduzierbarkeit: Ja, alle Schritte sind klar und umsetzbar.

Von Prozessor schrittweise ausführbar: Ja, ein Mensch kann sie auf Papier durchführen.

Nur Elementaroperationen (oder in solche übersetzbar): Ja, für alle per Definition.

Endliche Beschreibung: Ja, per der Beschreibungstext ist endlich.

Terminiert: Nein, es handelt sich um eine Endlosschleife. Die Werte von x verhalten sich $1 \rightarrow 2 \rightarrow 1 \rightarrow 2 \rightarrow \dots$, damit ist nie $x \geq 0$.

Aufgabe 3: Algorithmenkonstruktion

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen Tagespreise eines Tablets über die letzten Monate liefert. Sie möchten nun den maximalen prozentualen Preisanstieg zwischen zwei aufeinanderfolgenden Tagen für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer **Liste von Tagespreisen** des Tablets den **maximalen prozentualen Preisanstieg** von **zwei aufeinanderfolgenden Tagen** berechnet.

Liste von Tagespreise
 $(p_1, \dots, p_n)_{n \in \mathbb{N}}$



Max. % Preisanstieg
 Δ_{\max}

Aufgabe 3

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen **Tagespreise** eines Tablets über die letzten Monate liefert. Sie möchten nun den **maximalen prozentualen Preisanstieg** zwischen **zwei aufeinanderfolgenden Tagen** für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer **Liste von Tagespreisen** des Tablets den maximalen prozentualen Preisanstieg von **zwei aufeinanderfolgenden Tagen** berechnet.

Begriffe:

Tagespreisliste

zwei auff. Tage

max. % Preisanstieg

Tagespreis: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.

Tagespreisliste: chronologisch sortierte Liste an Tagespreisen $[p_1, \dots, p_n]$ mit $n \in \mathbb{N}_1$.

Tag: natürlicher Index in Tagespreisliste.

Zwei aufeinanderfolgende Tage: zwei Listenindizes: i und $i + 1$ (wobei $i, i + 1 < n$).

Preisanstieg: positive Veränderung einer reellen Zahl (dem Tagespreis): $p_{i+1} - p_i > 0$.

Maximaler prozentualer Preisanstieg: größter relativer Preisanstieg: $\max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i} \cdot 100$.

Wichtig: Wir sind hier nicht in einer Programmiersprache: Typen wie „`int`“ gibt es nicht \implies mathematische Notation benutzen oder eigenen Typ definieren (wie Tupel, ...)!

Liste von Tagespreise

$(p_1, \dots, p_n)_{n \in \mathbb{N}}$



Max. % Preisanstieg

Δ_{\max}

Gegeben: Chronologisch sortierte Liste $[p_1, \dots, p_n]$ positiver reeller Zahlen. Annahme: $n > 1$ („letzte Monate“).

Gesucht: Positive reelle Zahl $m = 100 \cdot \max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i}$ ($X \% \hat{=} \frac{X}{100}$).

- (1) Setze Δ_{\max} auf $-\infty$.
- (2) Setze i auf 1.
- (3) Solange ($i < n$), wiederhole:
 Setze test auf $(p_{i+1} - p_i)/p_i$.
 Wenn ($\text{test} > \Delta_{\max}$): Setze Δ_{\max} auf test.
 Erhöhe i um 1.
- (4) Ergebnis ist $100 \cdot \Delta_{\max}$.

$\Delta_{\max} = -\infty$ ist natürlich unpraktisch. Hier könnte man für $\Delta_{\max} = 0$ argumentieren (da „Anstieg“). Wir können aber auch die erste Differenz als Maximum setzen! Das kommt später.



- (1) $\Delta_{\max} \leftarrow -\infty$.
- (2) $i \leftarrow 1$.
- (3) Solange $i < n$:
 $\text{test} \leftarrow (p_{i+1} - p_i)/p_i$.
 Wenn $\text{test} > \Delta_{\max}$: $\Delta_{\max} \leftarrow \text{test}$.
 $i \leftarrow i + 1$.
- (4) Ergebnis ist $100 \cdot \Delta_{\max}$.

1. Eine formale, vollständige Induktion, oder:
2. „Textbasiert“: i wächst streng monoton an, die Schleife wird damit genau $n - 1$ mal durchlaufen (n ist konstant). Alle mathematischen Berechnungen terminieren per Konstruktion, ebenso die Zuweisungen. Der Algorithmus *terminiert*.

Zudem ist er *partiell korrekt*. Die maximale prozentuale Änderung ist immer die größte relative Differenz für alle $[p_1, \dots, p_{i+1}]$ im i -ten Durchlauf. Nach $n - 1$ Durchläufen: $[p_1, \dots, p_n]$. Basisfall mit $i = 2$, für Schritt $i \rightarrow i + 1$ (analog zum Maximumsalgorithmus).

- › Totale Korrektheit erfordert zwei Komponenten!

Terminiertheit: Der Algorithmus terminiert für jede definierte Eingabe.

Partielle Korrektheit: Wenn der Algorithmus für eine definierte Eingabe terminiert, ist das Ergebnis korrekt.

- › Formulierungen wie folgt reichen nicht aus:

- „Maximum ist sicher größer als alle betrachteten Alternativen“.

Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.

- „Der Algorithmus findet das gesuchte Ergebnis“.

Wir Beweisen zwar nicht formal, dennoch sollte ein ausreichendes Verständnis fürs Beweisen gezeigt werden.

Verifikation mit Induktion

- (1) Setze Δ_{\max} auf $-\infty$.
- (2) Setze i auf 1.
- (3) Solange ($i < n$), wiederhole:
Setze test auf $(p_{i+1} - p_i)/p_i$.
Wenn ($\text{test} > \Delta_{\max}$): Setze Δ_{\max} auf test .
Erhöhe i um 1.
- (4) Ergebnis ist $100 \cdot \Delta_{\max}$.

Wir zeigen nur die partielle Korrektheit („Verifikation“).

Für ein beliebiges aber festes $n \in \mathbb{N}$ mit $n \geq 2$ und der Eingabe p_1, \dots, p_n zeigen wir per vollständiger Induktion über i , dass in jedem Schritte gelte:

$$\Delta_{\max} \geq M(i) \quad M(i) = \max_{1 \leq j \leq i} \left\{ \frac{(p_{j+1} - p_j)}{p_j} \right\}$$

IA: Mit $i = 1$ ist $\Delta_{\max} = (p_2 - p_1)/p_1$ das triviale Maximum.

IH: Die Aussage $\Delta_{\max} \geq M(i)$ gilt für ein i .

IS: Es gilt $\Delta_{\max} \geq M(i)$ für i (IH). Wir zeigen, dass auch $\Delta_{\max} \geq M(i + 1)$ gilt:

1. $\text{test} > \Delta_{\max}$: Durch die Bedingung wird Δ_{\max} aktualisiert. Es gilt: $\Delta_{\max} = \text{test} \geq M(i + 1)$.
2. $\text{test} \leq \Delta_{\max}$: Der Anstieg durch $i + 1$ ist kein neues Maximum. Es gilt: $\Delta_{\max} \geq M(i + 1) \geq \text{test}$.

Nach $n - 1$ Schritten (durch $i < n$), werden alle p_1, \dots, p_n betrachtet $\implies \Delta_{\max}$ ist Maximum aller.

Textbasierte Aufwandsanalyse

- (1) Setze Δ_{\max} auf $-\infty$.
- (2) Setze i auf 1.
- (3) Solange ($i < n$), wiederhole:
 Setze test auf $(p_{i+1} - p_i)/p_i$.
 Wenn ($\text{test} > \Delta_{\max}$): Setze Δ_{\max} auf test.
 Erhöhe i um 1.
- (4) Ergebnis ist $100 \cdot \Delta_{\max}$.

Ob Division und Subtraktion als Elementaroperation zählen, kommt auf die Definition an.

- > Die 1. Zuweisungen 1 & 2 sind 2 Elementaroperationen.
- > Die äußere Schleife in 3 wird genau $n - 1$ mal durchlaufen.
 - Die 1. Schleifenanweisung hat 3 Elementaroperationen: Subtraktion, Division & Zuweisung.
 - 2. Schleifenanweisung ist 1 Vergleich und maximal 1 Zuweisung.
 - 3. Schleifenanweisung ist 1 Elementaroperation (je nach Definition auch 2).
- > Das Endergebnis ist eine Multiplikation (und je nach Definition 1 Zuweisung).
- > Im worst-case haben wir so: $2 + (n - 1) \cdot (3 + 2 + 1) + 2 = 6n - 2$ (also $\mathcal{O}(n)$).



Tabellarische Aufwandsanalyse

- (1) Setze Δ_{\max} auf $-\infty$.
- (2) Setze i auf 1.
- (3) Solange ($i < n$), wiederhole:
 Setze test auf $(p_{i+1} - p_i)/p_i$.
 Wenn ($\text{test} > \Delta_{\max}$): Setze Δ_{\max} auf test.
 Erhöhe i um 1.
- (4) Ergebnis ist $100 \cdot \Delta_{\max}$.

Anweisung	Aufwand	Wie oft?	Was?
Setze Δ_{\max} auf $-\infty$	1 E	1	←
Setze i auf 1	1 E	1	←
Solange ($i < n$), wiederhole	1 E	n	<
Setze test auf $(p_{i+1} - p_i)/p_i$	3 E	$n - 1$	-, ÷, ←
Wenn ($\text{test} > \Delta_{\max}$)	1 E	$n - 1$	>
Setze Δ_{\max} auf test	1 E	max. $n - 1$	←
Erhöhe i um 1	1 E	$n - 1$	++
Ergebnis ist $100 \cdot \Delta_{\max}$	2 E	1	·, →

- So kommen wir auf den selben worst-case:
 $1 E + 1 E + (n - 1) \cdot (3 E + 2 E + 1 E) + 2 E = (6n - 2) E$
- Der best-case führt die bedingte Anweisung nie aus:
 $1 E + 1 E + (n - 1) \cdot (3 E + 2 E + 0 E) + 2 E = (5n - 1) E$

Aktuell verschmelzen wir den Vergleich der for-Schleife mit dessen Inkrement (++).



Eine Alternative

- › Wir können erst alle Preisanstiege berechnen und dann darin das Maximum suchen:

given days $1, \dots, n$ **and** prices (p_1, \dots, p_n) **indexable by** p_i
when $n < 2$ **then output** "Ein Preisanstieg braucht mind. 2 Tage" **and stop**

Optional: Robust gegenüber zu wenig Tagen.

make list $L = (l_1, \dots, l_{n-1})$ **of size** $n-1$ **indexable by** l_i
for each d **from** 1 **to inclusive** $n-1$ **in steps of** 1
 • **let** l_d **be** $(p_{d+1} - p_d) / p_d \cdot 100$ } Berechne den Preisanstieg an Tag d für alle Tage.

[Jetzt hält l_i den Preisanstieg an Tag i und hat $n-1 \geq 1$ Elemente]
let max_diff **be** l_1
for each diff **from** 2 **to inclusive** $n-1$ **in steps of** 1
 • **when** $\text{diff} > \text{max_diff}$ **then let** max_diff **be** diff } Normale Suche nach dem Maximum.

output "Der Maximale Preisanstieg ist diff "

Aussicht: Übungsblatt 2

- › Der *Algorithmenentwurf* ist nur ein Schritt der Algorithmenkonstruktion
- › Annahmen festhalten
- › Auch die Präzedenzregeln sind sicherlich hilfreich. Die wichtigsten:

$a++$, $a--$	\longrightarrow	$!a$, $-a$, $++a$, $--a$	\longrightarrow	$a * b$, a / b , $a \% b$
	\longrightarrow	$a + b$, $a - b$	\longrightarrow	$a == b$, $a < b$, ...
	\longrightarrow	$a \wedge b$	\longrightarrow	$a \&\& b$
	\longrightarrow	$a \parallel b$		

*The White Rabbit put on his spectacles. „Where shall I begin, please your Majesty?“ he asked. „Begin at the beginning,“ the King said gravely, „and go on till you come to the end: then stop.“
– Lewis Carroll [Car65, chp. 12].*

Abschließendes

- [Amd67] Gene Amdahl. *Amdahl's law*. 1967
- [Car65] Lewis Carroll. *Alice's Adventures in Wonderland*. Nov. 1865
- [EpiAD] Epictetus. *Discourses, Book I*. 108 AD

- Wir haben die Geburt von Variablen sowie ihre Wachstumsphase kennengelernt:
 - Die *Deklaration* (`int x`) reserviert einen Namen samt Charakteristika (Typ, ...)
 - Die *Initialisierung* (`int x; x = 5`) beschreibt die erste Wertzuweisung
 - Eine (*Wert-*)*Zuweisung* (`x = 3`) ist jede weitere Änderung des gespeicherten Wertes
- In die Wahl des „richtigen“TM Datentyps fließen viele Informationen mit ein.

