

252-0027

Einführung in die Programmierung

Übungen

Woche 2: Eclipse und EBNF

Timo Baumberger

Departement Informatik


ETH Zürich

Organisatorisches



- Mein Name: Timo Baumberger
- Bei Fragen: tbaumberger@student.ethz.ch
(*Discord: troxhi*)
 - Mails bitte mit «[EProg25]» im Betreff
- Meine Website: timobaumberger.com
- Neue Aufgaben: **Dienstag Abend** (im Normalfall)
- Abgabe der Übungen bis **Dienstag Abend (23:59)** Folgewoche
 - Abgabe immer via Git
 - Lösungen in separatem Projekt auf Git

Inhalt

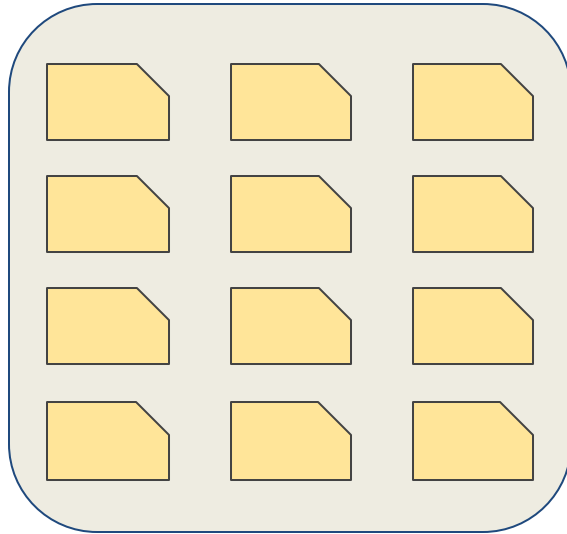
- **Git Einführung**
- **EBNF Beispiele**
- **EBNF Rekursion**
- **Ableitungsbaum / Ableitungstabelle**
- **EBNF Aufgaben**
- **Java** 

Was ist Git?

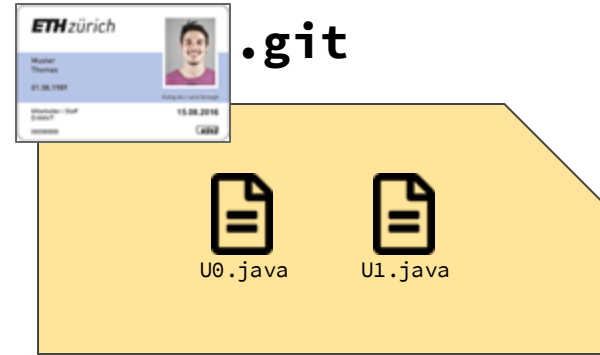


- **Analogie: Google Drive / Dropbox / Polybox**
- **Git Commit ~ Lokale Änderung**
- **Git Push ~ Datei hochladen**
- **Git Pull ~ Datei herunterladen**
- **Git bietet (zusätzlich) Versionierung an**

Git Repository

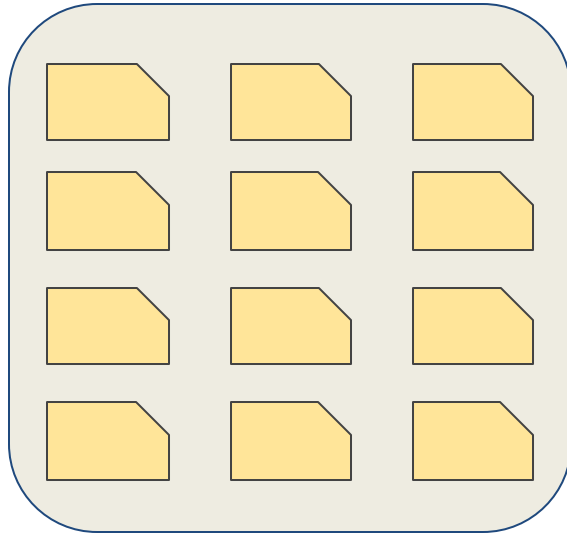


ETH Git-Server

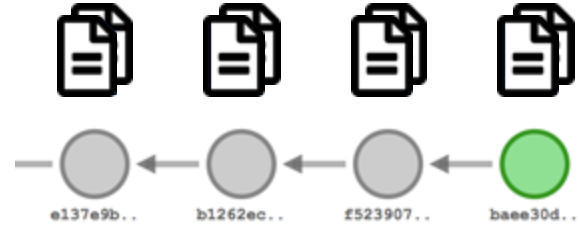


Jedes Repository auf dem Git-Server ist privat

Git Repository



ETH Git-Server



Ältester Commit

Neuester Commit

Jedes Repository auf dem Git-Server
Enthält eine Folge von **Commits** (die **History**)



.git

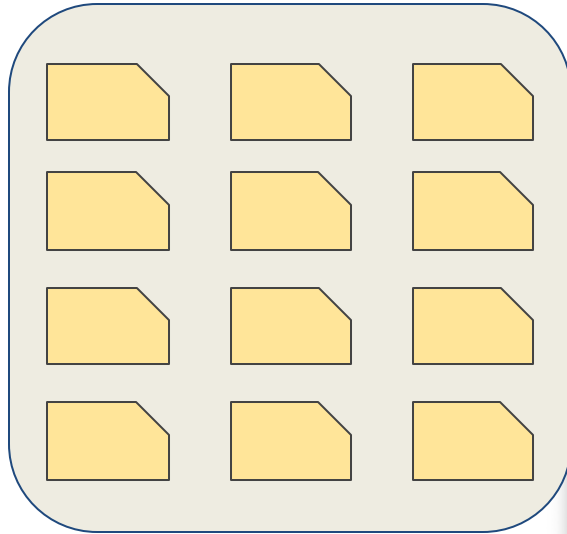


U0.java



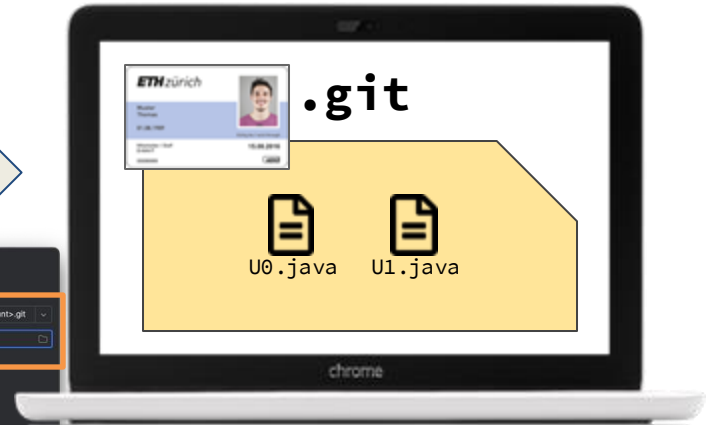
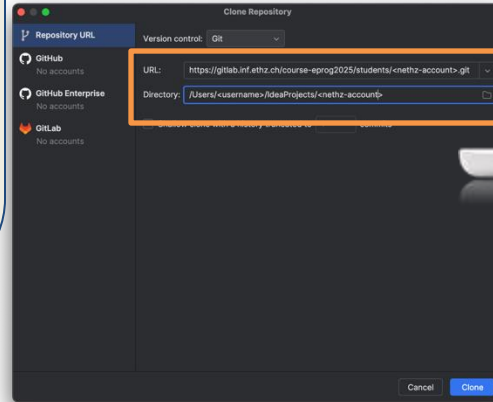
U1.java

Git Clone: Einmaliges Einrichten



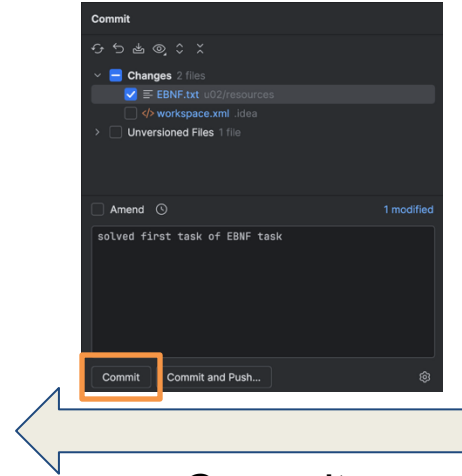
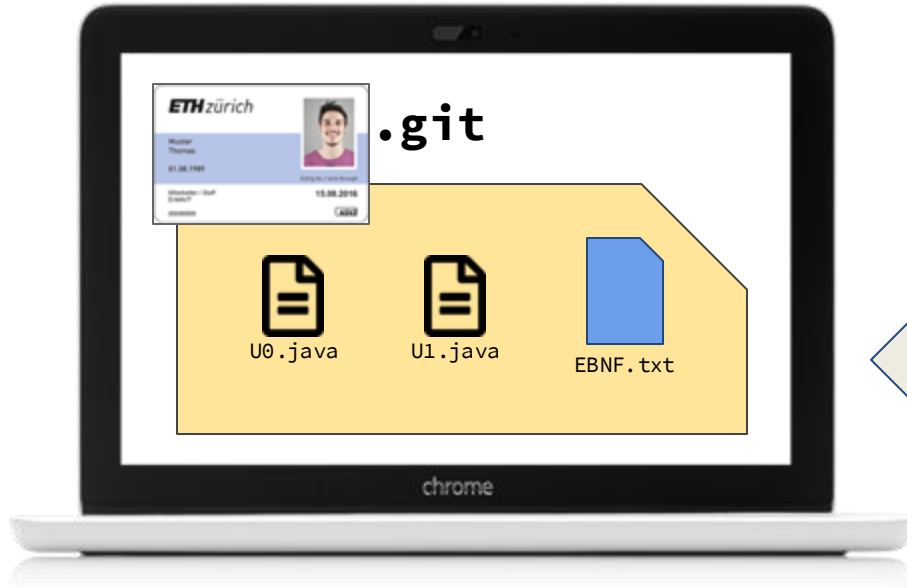
ETH Git-Server

Clone
Kopiert das ganze Repository
auf den eigenen Computer



Lokales Git-Repository

Git Commit: Fortschritt speichern



Commit

Fügt neuen Commit mit
Änderungen/neuen Dateien
der *lokalen* History hinzu



EBNF.txt

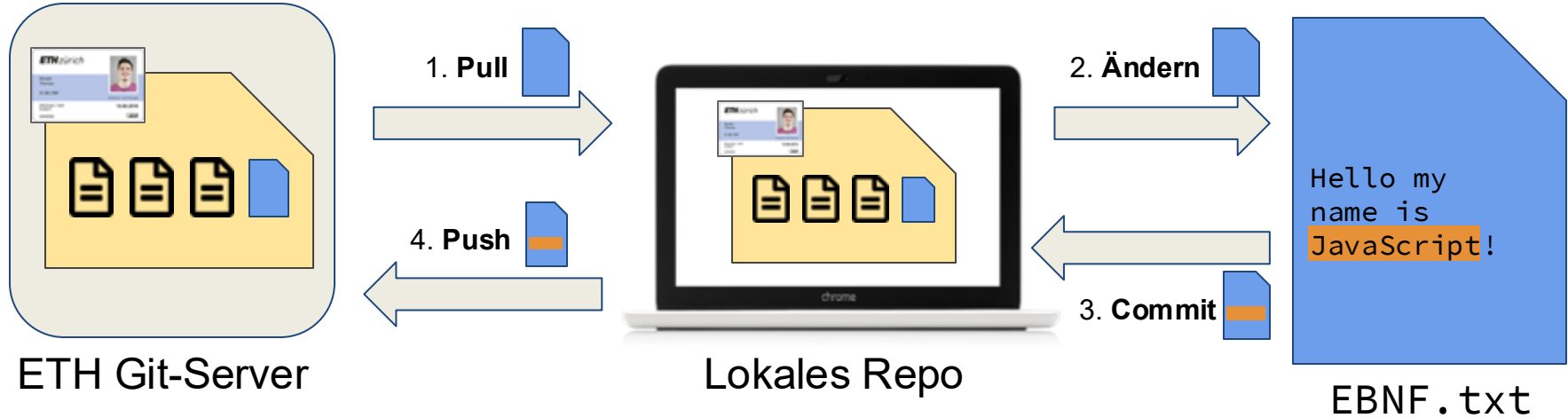
Git Push: Abgeben



Git Pull: Aufgaben / Feedback laden



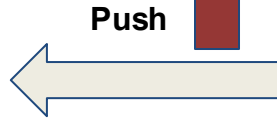
Git Pull/Push-Workflow



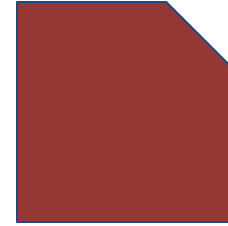
Git Repository ändert sich!



ETH Git-Server

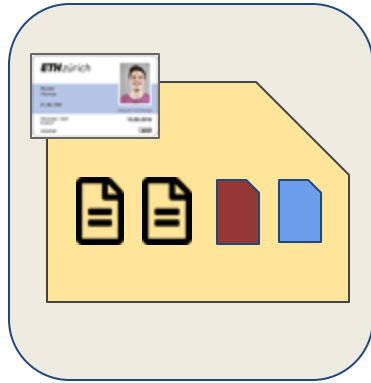


Lokales Repo
(von eurem TA)

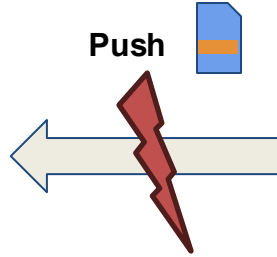


Feedback.txt

Git Merge Conflict



ETH Git-Server

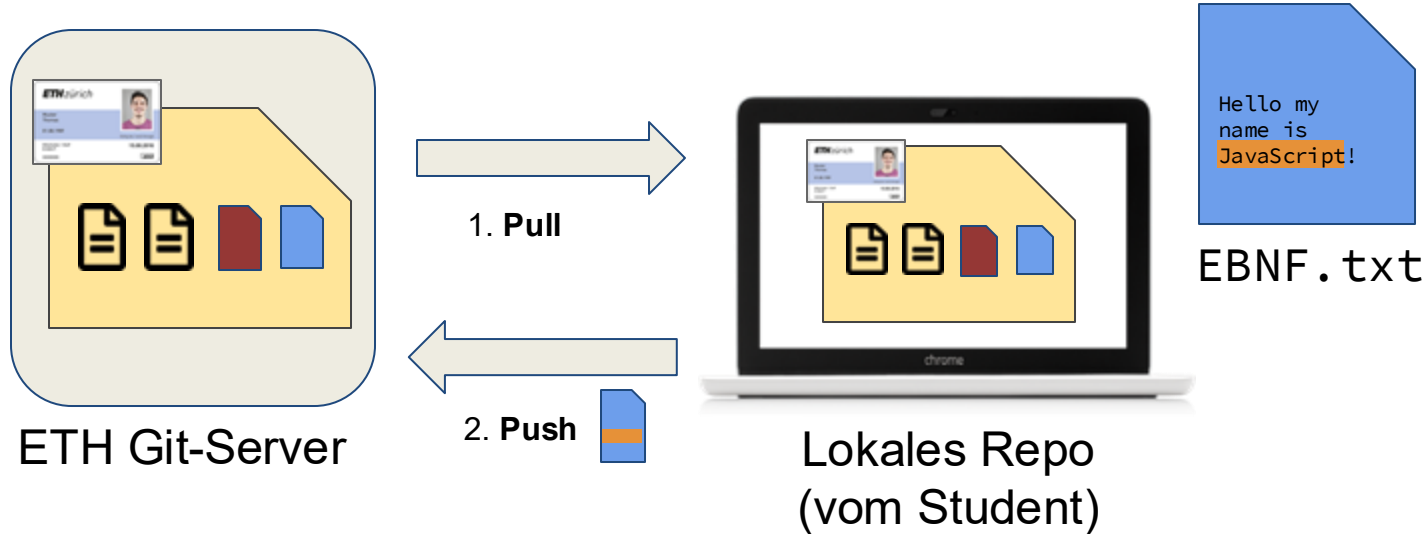


Lokales Repo
(vom Student)



EBNF.txt

Git Merge Conflict verhindern



Immer zuerst pull dann push!

Interesse an Git geweckt?

- 🔗 Fortgeschrittene Konzepte können hier trainiert werden (<https://learngitbranching.js.org/>)
- 🔗 Allgemeine Git Dokumentation (<https://git-scm.com/docs/user-manual>)

Git: Clone, Aus- und Einchecken



(Zusammen)

Authentifizierung: Token (HTTP) oder SSH

- Token müsst ihr bei GitLab generieren und in IntelliJ angeben
- SSH Keypair könnt ihr bei euch lokal erstellen
 - Private Key **NIE** weitergeben
 - Public Key bei GitLab hinterlegen
 - Nutzt den Default-File-Namen für die Keys

Siehe: <https://docs.gitlab.com/user/ssh/>

Häufige Git-Fehler vermeiden

- ❌ Vor erster Benutzung auf gitlab.inf.ethz.ch anmelden
- ❌ In U02 Aufgabe 4, <nethz-account> im Link inklusive <> ersetzen
- ❌ In U02 Aufgabe 4, bei Problemen Link per Hand abtippen
- ❌ Files/Directories/Projects/... immer in IntelliJ löschen bzw. ändern (nie im File Explorer/Finder)
- ❌ Immer Pull und dann Push
- ❌ “Commit und Push” statt nur “Commit”
 - ✂ Falls Sie auf Commit geklickt haben, drücken Sie manuell per Rechtsklick auf das Repository und wählen Sie “Push” oder “Push to origin” aus
- ❌ Importieren bei neuen Projects nicht vergessen!

“Ok, was soll ich mir merken???”



- Bevor ihr Änderungen vernehmt: Git Pull
- Bevor ihr eure Änderungen commiten wollt: Git Pull
- Sobald ihr Änderungen gemacht habt: Git Commit und Git Push

EBNF – Einfaches Beispiel – Bahnhof



EBNF – Einfaches Beispiel – Bahnhof



EBNF – Einfaches Beispiel – Bahnhof



EBNF-Beschreibung von Anzeigetafel

<zugbezeichnung> <= IC61 | RE | S3 | EC

<abfahrtszeit> <= <stunde> : <minute>

<zielbahnhof> <= Zürich HB | Bern | Basel | Wil | Chur | Interlaken Ost

<stunde> = ?

<minute> = ?

<anzeigetafel> <= <zugbezeichnung> <abfahrtszeit> <zielbahnhof>

EBNF – Beispiel Programmiersprache

```
<programm> <= PROGRAM <bezeichner>
      BEGIN
      { <zuweisung> ; }
      END
<zuweisung> <= <bezeichner> := ( (<zahl>|<bezeichner>) |<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

Wie prüfe ich, was legal ist?

1. Informeller Beweis
2. Tabellen
3. Ableitungsbaum
4. Graphische Darstellung

JETZT

```

<programm> <= PROGRAM <bezeichner>
      BEGIN
        { <zuweisung> ; }
      END
<zuweisung> <= <bezeichner> := ( (<zahl>|<bezeichner>) |<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9

```

PROGRAM DEMO1

BEGIN

A0:=3;

B:=+45;

H:=-100023;

C:=A;

D123:=B34A;

ESEL:=GIRAFFE;

TEXTZEILE:="HALLO";

END


```

<programm> <= PROGRAM <bezeichner>
      BEGIN
        { <zuweisung> ; }
      END
<zuweisung> <= <bezeichner> := ( (<zahl>|<bezeichner>) |<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9

```

PROGRAM DEMO1
BEGIN

```

A0:=3;
B:=+45;
H:=-100023;
C:=A;
D123:=B34A;
ESEL:=GIRAFFE;
TEXTZEILE:="HALLO";

```

END



```
<programm> <= PROGRAM <bezeichner>
      BEGIN
        { <zuweisung> ; }
      END
<zuweisung> <= <bezeichner> := ((<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

PROGRAM DEMO2

BEGIN

myVariable3 := 5

2GOOD2BETRUE := 6ER

END

```

<programm> <= PROGRAM <bezeichner>
      BEGIN
        { <zuweisung> ; }
      END
<zuweisung> <= <bezeichner> := ((<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9

```

PROGRAM DEMO2

BEGIN

myVariable3 := 5

2GOOD2BETRUE := 6ER

END



```
<programm> <= PROGRAM <bezeichner>
      BEGIN
        { <zuweisung> ; }
      END
<zuweisung> <= <bezeichner> := ( (<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

PROGRAM DEMO3

BEGIN

GOODNAME := +-5

MARK := 5.5

END

```
<programm> <= PROGRAM <bezeichner>
      BEGIN
        { <zuweisung> ; }
      END
<zuweisung> <= <bezeichner> := ( (<zahl>|<bezeichner>)|<string>)
<bezeichner> <= <buchstabe> {<buchstabe>|<ziffer>}
<zahl> <= [-|+] <ziffer> {<ziffer>}
<string> <= "{<buchstabe>|<ziffer>}"
<buchstabe> <= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<ziffer> <= 0|1|2|3|4|5|6|7|8|9
```

PROGRAM DEMO3

BEGIN

GOODNAME := +-5

MARK := 5.5

END



Rekursion

Rekursion ist, wenn sich eine Regel selbst aufruft bzw. definiert.

Wir unterscheiden dabei zwischen direkter und indirekter Rekursion.



Beispiel:

$\langle A \rangle \leq \langle A \rangle \mid a$

Beispiel:

$\langle A \rangle \leq \langle B \rangle \mid a$
 $\langle B \rangle \leq b \langle A \rangle$

Sinnvolle/endliche Rekursion braucht Abbruchsoption (“base case”)

Rekursion Beispiele

EBNF-Beschreibung von List

$\langle \text{letter} \rangle \leq a \mid b \mid c$

$\langle \text{list} \rangle \leq (\langle \text{letter} \rangle , \langle \text{list} \rangle) \mid \langle \text{letter} \rangle$

Legal?
a
a,b
a,b,c
a,a,a,a,a,a,a,a,a
a,bb,c

Rekursion Beispiele

Erstellen Sie eine Grammatik, die eine einfache Dateiverzeichnisstruktur beschreibt. In dieser Struktur besteht ein Verzeichnis entweder aus einem einzelnen Ordner oder aus einer Hierarchie von Ordnern, die durch Schrägstriche (/) getrennt sind.

Die folgenden Ordnernamen sind zulässig: dokumente, bilder, musik, downloads, desktop.

Das Verzeichnis kann aus beliebig vielen verschachtelten Ordnern bestehen. Beispielhafte gültige Verzeichnisse sind:

- dokumente
- bilder/musik
- downloads/bilder/dokumente

Repetition: Ableitungen

- Ableitungstabelle
 - Erste Zeile ist Startregel
 - Letzte Zeile ist Zeichenfolge
 - Übergang zwischen zwei Zeilen entspricht Ableitungsschritt
- Ableitungsbaum
 - Wurzel ist Namen der Startregel
 - Blätter sind Zeichen
 - Verbindungen stehen für einen Ableitungsschritt

Besipiel: Ableitung von c,b,a als Baum

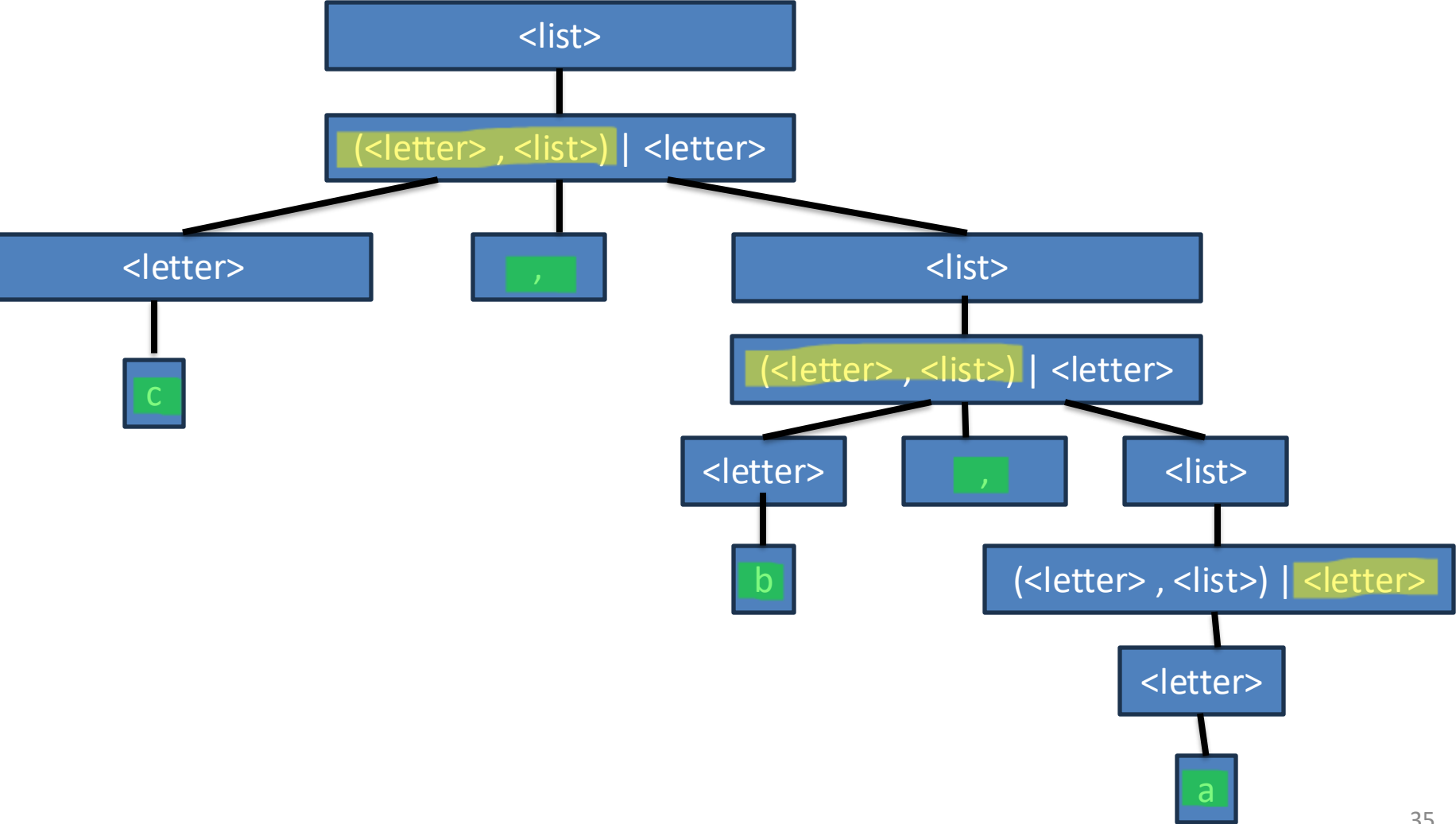
EBNF-Beschreibung von List

$\langle \text{letter} \rangle \leq a \mid b \mid c$

(R1)

$\langle \text{list} \rangle \leq (\langle \text{letter} \rangle , \langle \text{list} \rangle) \mid \langle \text{letter} \rangle$

(R2)



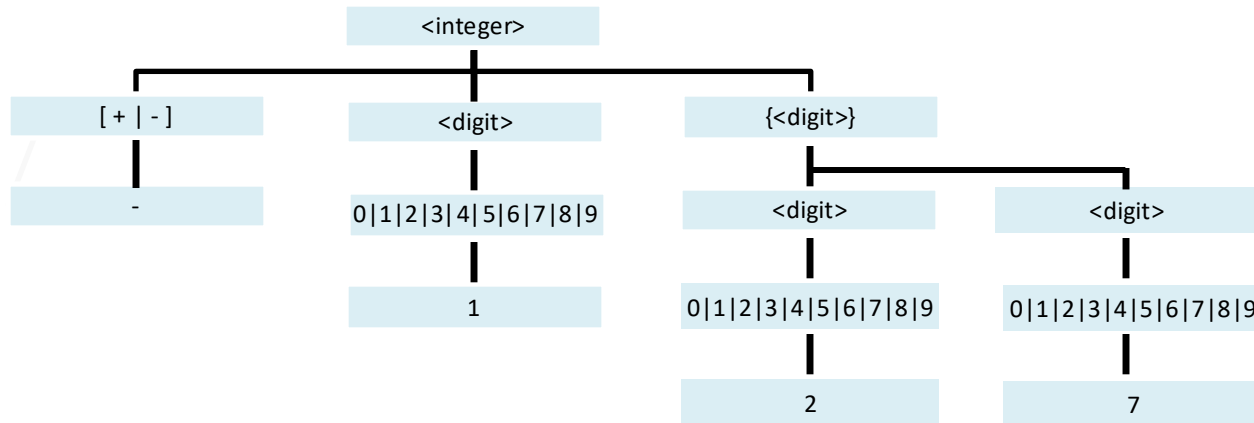
Beispiel: Ableitung von -127 als Baum

<digit>	←	0 1 2 3 4 5 6 7 8 9
<integer>	←	[+ -] <digit> {<digit>}

Beispiel: Ableitung von -127 als Baum

$\langle \text{digit} \rangle \leftarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$\langle \text{integer} \rangle \leftarrow [+ \mid -] \langle \text{digit} \rangle \{ \langle \text{digit} \rangle \}$



Beispiel: Ableitung von -127 als Tabelle

(R1) <digit> ← 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

(R2) <integer> ← [+ | -] <digit> {<digit>}

Beispiel: Ableitung von -127 als Tabelle

(R1) <digit> ← 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

(R2) <integer> ← [+ | -] <digit> {<digit>}

<integer> ← [+ | -] <digit> {<digit>}

 ← + | - <digit> {<digit>}

 ← - <digit> {<digit>}

 ← - <digit> <digit> <digit>

 ← - 1 <digit> <digit>

 ← - 1 2 <digit>

 ← - 1 2 7

(R2)

Option gewählt

- gewählt

2 mal wiederholt

(R1) und 1 gewählt

(R1) und 2 gewählt

(R1) und 7 gewählt

EBNF Notation

- In alten Prüfungen wird oft kursiv verwendet für EBNF-Regeln.
- *digit* statt <digit>
- Ab diesem Semester ist eine EBNF-Regel nur korrekt, wenn sie durch < > gekennzeichnet ist.
- Ebenfalls verwenden wir <- statt <=, beides wird aber als korrekt bewertet.

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

- | | |
|---------------------|--------------------|
| ■ 2023 | ■ 12cd.34 |
| ■ _2023 | ■ 09.2023 |
| ■ Back | ■ face |
| ■ 1a2b3c | ■ 123k |
| ■ 1a2b3k | ■ 0010K |
| ■ 27_09_2023 | ■ 27_9.2023 |
| ■ 09.2023k | ■ 27.9.2023 |

<i>digit</i>	\Leftarrow	$\boxed{0} \mid \boxed{1} \mid \boxed{2} \mid \boxed{3} \mid \boxed{4} \mid \boxed{5} \mid \boxed{6} \mid \boxed{7} \mid \boxed{8} \mid \boxed{9}$
<i>separator</i>	\Leftarrow	$\boxed{-}$
<i>buchst</i>	\Leftarrow	$\boxed{A} \mid \boxed{B} \mid \boxed{C} \mid \boxed{D} \mid \boxed{E} \mid \boxed{F} \mid \boxed{a} \mid \boxed{b} \mid \boxed{c} \mid \boxed{d} \mid \boxed{e} \mid \boxed{f}$
<i>zahl</i>	\Leftarrow	$\text{digit} \{ [\text{separator}] \text{digit} \}$
<i>int</i>	\Leftarrow	$\text{digit} \{ \text{digit} \}$
<i>real</i>	\Leftarrow	$\text{digit} \{ \text{digit} \} [\boxed{.} \text{digit} \{ \text{digit} \}]$
<i>bd</i>	\Leftarrow	$\text{buchst} \mid \text{digit}$
<i>mix1</i>	\Leftarrow	$\text{bd} \{ \text{bd} \}$
<i>mix2</i>	\Leftarrow	$\text{digit} \{ \text{digit} \} \boxed{k}$
<i>mix</i>	\Leftarrow	$\text{mix1} \mid \text{mix2}$
<i>value</i>	\Leftarrow	$\text{zahl} \mid \text{real} \mid \text{int} \mid \text{mix}$

Kahoot

<https://create.kahoot.it/share/eprog-u2-baumberger/b5c6cdf1-36e8-40c3-b5d5-aa4153e00fb1>

<https://create.kahoot.it/share/eprog-u2-zusatz/e953915f-7865-4bdc-b357-9117def8cf3f>

<https://create.kahoot.it/share/duplikat-von-u2-ebnf-schwer/dcc4a55b-915a-4705-8bf1-c1c1866ece51> **(schwieriger)**

Zusatzaufgaben

- Erstellen Sie eine Beschreibung `<palindrom>`, welche als legale Symbole alle Zahlen zulässt, die von Vorne und Hinten gleich gelesen werden und die nur die Ziffern von 1 bis 4 verwenden. Beispiele sind 11, 232, 444
- Erstellen Sie eine Beschreibung `<five>`, welche alle Summen von positiven Zahlen zulässt, welche 5 ergeben. Beispiele sind "1 + 4", "2 + 1 + 1 + 1", "5"
- Erstellen Sie eine Beschreibung `<oddEight>`, , die alle Zahlen enthält, in denen die Ziffer 8 ungerade oft vorkommt. Beispiele sind 8, 128, 8881

Zusatzaufgabe 1

Erstellen Sie eine Beschreibung $\langle \text{palindrom} \rangle$, welche als legale Symbole alle Zahlen zulässt, die von Vorne und Hinten gleich gelesen werden und die nur die Ziffern von 1 bis 4 verwenden. Beispiele sind 11, 232, 444

$\langle \text{palindrom} \rangle \Leftarrow [1\langle \text{palindrom} \rangle 1 \mid 2\langle \text{palindrom} \rangle 2 \mid 3\langle \text{palindrom} \rangle 3 \mid 4\langle \text{palindrom} \rangle 4 \mid 1 \mid 2 \mid 3 \mid 4]$

Zusatzaufgabe 2

Erstellen Sie eine Beschreibung `<five>`, welche alle Summen von positiven Zahlen zulässt, welche 5 ergeben. Beispiele sind “1 + 4”, “2 + 1 + 1 + 1”, “5”

```
<five>  <= 0 + <five> | 1 + <four> | 2 + <three> | 3 + <two> | 4 + <one> | 5 + <zero> | 5
<four>  <=          0 + <four> | 1 + <three> | 2 + <two> | 3 + <one> | 4 + <zero> | 4
<three> <=                0 + <three> | 1 + <two> | 2 + <one> | 3 + <zero> | 3
<two>   <=                      0 + <two> | 1 + <one> | 2 + <zero> | 2
<one>   <=                            0 + <one> | 1 + <zero> | 0
<zero>  <=                                  0 + <zero> | 0
```

Zusatzaufgabe 3

Erstellen Sie eine Beschreibung `<oddEight>`, , die alle Zahlen enthält, in denen die Ziffer 8 ungerade oft vorkommt. Beispiele sind 8, 128, 8881

```
<oddEight>      <= [+|-] <oddEight2>
<oddEight2>     <= [<evenEight>]      8 [<trailingEvenEight>]
<trailingOddEight> <= [<trailingEvenEight>] 8 [<trailingEvenEight>]
<evenEight>      <= <noEight> | <oddEight2><trailingOddEight>
<trailingEvenEight> <= <trailingNoEight> | <trailingOddEight><trailingOddEight>
<noEight>        <= (1 | 2 | 3 | 4 | 5 | 6 | 7 | 9) [<trailingNoEight>]
<trailingNoEight> <= (0 | <noEight>) [<trailingNoEight>]
```

Java



- Zur Entwicklung wird die Java JDK (Java Development Kit) verwendet
- Enthält Compiler, Libraries, wird zur Entwicklung von Programmen verwendet
- Java JRE (Java Runtime Environment) wird von “Java-Nutzern” verwendet
- Java JRE wird von 3 Milliarden Geräten verwendet

Java Performance

- **Java Performance wird SEHR oft verkannt (falsch beurteilt)**
- **Sehr gut im Vergleich zu anderen Programmiersprachen wie Go, C++, Rust**
- **Braucht viel Arbeitsspeicher (Memory)**
- **Java wird stetig weiterentwickelt und verbessert**
- **Seit GraalVM Native Image deutlich besser**