

Computergestützter Sprachvergleich mit Python und JavaScript

Johann-Mattis List
Centre des Recherches sur l'Asie Orientale
Paris

Sommersemester 2015
Heinrich-Heine Universität Düsseldorf

Contents

1 Einführung	5
1.1 Organisatorisches	5
1.1.1 Zeiten	5
1.1.2 Seminarplan	5
1.1.3 "Wie kriege ich einen BN?"	5
1.1.4 "Wie kriege ich einen AP?"	5
1.1.5 "Wie wird die Klausur aussehen?"	6
1.1.6 "Wie soll ich mich auf die Klausur vorbereiten?"	6
1.1.7 Seminarwebsite	6
1.2 Vorstellung	6
1.2.1 ... meiner Person	6
1.2.2 ... des Seminarthemas	7
1.2.3 ... des Seminarthemas	7
1.3 Allgemeines	11
1.3.1 ... zum Programmieren	11
1.3.2 ... zum Programmieren in der Linguistik	12
1.4 Spezielles	12
1.4.1 ... zu Algorithmen, Skripten und Programmen	12
1.4.2 ... zur Grundausstattung fürs Programmieren	13
1.4.3 ... zu Python und JavaScript	14
2 Erste Schritte in Python	15
2.1 Allgemeines zu Python	15
2.1.1 Herkunft	15
2.1.2 Charakteristik	15
2.1.3 Installation	15
2.2 Bibliotheken und Entwickertools	16
2.2.1 Allgemeines	16
2.2.2 Empfohlene Python-Bibliotheken	16
2.2.3 Empfohlene Entwickertools	16

2.3 Ein erstes Programmierbeispiel	17
2.3.1 Das Problem	17
2.3.2 Der Algorithmus	17
2.3.3 Die Python-Implementierung	18
3 Erste Schritte in JavaScript	20
3.1 Allgemeines zu JavaScript	20
3.1.1 Herkunft	20
3.1.2 Charakteristik	21
3.1.3 Installation	21
3.2 Bibliotheken und Entwicklertools	21
3.2.1 Webbrowser	21
3.2.2 Bibliotheken	21
3.2.3 Entwicklertools	22
3.3 Ein erstes Programmierbeispiel	22
3.3.1 Die Kölner Phonetik in JavaScript	22
4 Datentypen und Variablen	24
4.1 Variablen	24
4.1.1 ... im Allgemeinen	24
4.1.2 ... in Python	25
4.1.3 ... in JavaScript	26
4.2 Datentypen	27
4.2.1 ... im Allgemeinen	27
4.2.2 ... in Python	28
4.2.3 ... in JavaScript	29
4.3 Praktische Beispiele	30
4.3.1 Die Dreiballkaskade	30
4.3.2 Die Dreiballkaskade in Python	31
4.3.3 Die Dreiballkaskade in Javascript	32
5 Operatoren und Kontrollstrukturen	34
5.1 Operatoren	34
5.1.1 Allgemeines	34
5.1.2 ... in Python	35
5.2 Kontrollstrukturen	42
5.2.1 Allgemeines	42
5.2.2 ...in Python	43
5.2.3 ... in JavaScript	46
6 Funktion	48
6.1 Funktionen im Allgemeinen	48
6.1.1 Begriffserklärung	48
6.1.2 Typen von Funktionen	48
6.2 Funktionen in Python	48
6.2.1 Grundlagen	49
6.2.2 Parameter und Schlüsselwörter	50
6.2.3 Spezialfälle	51
6.3 Funktionen in JavaScript	53
6.3.1 Grundlagen	53

6.3.2 Parameter und Schlüsselwörter	54
6.3.3 Spezialfälle	54
7 Sequenzvergleiche in der historischen Linguistik	55
7.1 Sprachwandel	55
7.1.1 Sprachen	55
7.1.2 Sprachwandel im Allgemeinen	58
7.1.3 Lautwandel	59
7.2 Lautwandel	60
7.2.1 Mechanismen des Lautwandels	60
7.2.2 Typen des Lautwandels	61
7.2.3 Typen des Lautwandels	62
7.3 Lautwandel	62
7.3.1 Typen des Lautwandels	62
7.3.2 Die komparative Methode	62
7.3.3 Die komparative Methode	63
7.4 Sequenzalinierung	65
7.4.1 Sequenzen und Sequenzvergleiche	65
7.4.2 Alinierung	67
7.4.3 Phonetische Alinierung	67
8 Sequenzalinierung in JavaScript	68
8.1 Automatische Sequenzalinierung	68
8.1.1 Das Problem	68
8.1.2 Die Lösungsstrategie	69
8.1.3 Der Algorithmus	69
8.2 Sequenzalinierung in JavaScript	70
8.2.1 Vorüberlegungen	70
8.2.2 Implementierung der Matrixberechnung	70
8.2.3 Implementierung des Traceback	72
8.3 Interaktive Sequenzalinierung	73
8.3.1 Grundlagen	73
8.3.2 Implementierung	73
8.3.3 Demo	74
9 Sequenzalinierung in Python	74
9.1 Sequenzalinierung in Python	74
9.1.1 Allgemeine Strategien	74
9.1.2 Implementierung der Matrixberechnung	75
9.1.3 Implementierung des Traceback	77
9.2 Lautklassenbasierte Alinierung	79
9.2.1 Noch mal zu den Lautklassen	79
9.2.2 Lautklassenkonvertierung mit Python	81
9.2.3 Implementierung	83
9.3 Phonetische Alinierung mit LingPy	84
9.3.1 Was ist LingPy?	84
9.3.2 Grundlegende Befehle	85
9.3.3 Workflows	86

10 Linguistischer Hintergrund zum Sprachvergleich	92
10.1 Phylogenetische Rekonstruktion	92
10.1.1 Innere und äußere Sprachgeschichte	92
10.1.2 Stammbäume und Wellen	93
10.1.3 Darstellung und Analyse	94
10.2 Lexikostatistik	95
10.2.1 Hintergrund und Grundannahmen	95
10.2.2 Praktische Umsetzung	95
10.2.3 Kritik	96
10.2.4 Komparanda	96
10.3 Computergestützte Rekonstruktion	96
10.3.1 Algorithmen	97
10.3.2 Formate	97
11 Automatischer Sprachvergleich mit Python	100
11.1 Automatischer Sprachvergleich	100
11.1.1 Sequenzdistanzen	100
11.1.2 Kognatenerkennung	100
11.1.3 Phylogenetische Rekonstruktion	101
11.2 Sprachvergleich mit LingPy	102
11.2.1 Eingabeformate	102
11.2.2 Analysen	104
11.2.3 Ausgabeformate	104
11.3 Workflows	105
11.3.1 Allgemeines vorweg	105
11.3.2 Kognatenerkennung mit LingPy	107
11.3.3 Integration mit externen Tools	108
12 Geoplots mit JavaScript und D3	110
12.1 Darstellung geographischer Daten	110
12.1.1 GeoJson und TopoJson	110
12.1.2 D3	110
12.2 Eine Beispielapplikation	110
12.2.1 Idee	110
12.2.2 Idee	111
12.2.3 Vorbereitung	111
12.2.4 Applikation	111

1 Einführung

1.1 Organisatorisches

1.1.1 Zeiten

Die Sitzungen werden an vier Tagen jeweils zu drei Zeiten abgehalten:

- I: 9 Uhr - 10:30
- II: 11 Uhr - 12:30
- III: 13:30 - 15:00

Zusätzlich gibt es noch für die restlichen Stunden der ersten drei Sitzungstage jeweils eine

- IV: Vertiefungsaufgabe

1.1.2 Seminarplan

Generelles Ziel des Seminars ist es, dass wir in den vier Tagen, die wir zur Verfügung haben, eine möglichst umfassende Einführung in die Thematik erhalten, die es allen Teilnehmern ermöglicht, bei Wunsch und Interesse, ihre Fähigkeiten auf dem Gebiet zu vertiefen. Von den vier Tagen, die uns bleiben, dienen die ersten beiden Tage zur

grundlegenden Einführung in die Thematik, während wir an den letzten beiden Tagen versuchen wollen, **konkrete Beispiele aus der Praxis** zu besprechen und umzusetzen.

Der vollständige Plan kann von der Seminarwebseite abgerufen werden und wurde auch vor dieser Sitzung bereits an alle Teilnehmer versandt.

1.1.3 "Wie kriege ich einen BN?"

Da, soweit ich das verstanden habe, für einen BN lediglich eine Teilnahme erforderlich ist und keine weiteren Leistungen vom Dozenten eingefordert werden dürfen, bekommen alle einen BN, die sich bei mir melden und mit den erforderlichen Angaben in eine entsprechende Liste eintragen. Die Liste selbst wird zwei Mal im Laufe der Sitzungstage ausgegeben. Diejenigen, die zu keiner dieser Sitzungen erscheinen und sich als BN-Anwärter eintragen, können sich nachher noch persönlich an mich wenden. Ich behalte mir jedoch vor, Kandidaten, die ich kein einziges Mal im Seminar gesehen habe, den BN zu verweigern (wir müssen es ja nicht übertreiben...).

1.1.4 "Wie kriege ich einen AP?"

Bitte bringen Sie die erforderlichen Unterlagen zu einer der Seminarsitzungen mit, damit ich sie unterschreiben kann. Es wird voraussichtlich nur die Möglichkeit geben, eine Klausur zu schreiben. In ganz dringenden Fällen können wir auch über eine Hausarbeit reden (allerdings ist das sehr schwierig, eine Hausarbeit in diesem Bereich zu schreiben!), wobei ich mir immer vorbehalte, das abzulehnen. Aller Voraussicht nach schreiben wir die Klausur am 18.

September zwischen 14 und 16 Uhr. Eventuell teile ich die Termine in zwei Termine auf, einen für die Bachelor- und einen für die Masterkandidaten. In diesem Fall kommt auch der 11. September (gleiche Uhrzeit) als Termin in Frage.

1.1.5 “Wie wird die Klausur aussehen?”

Die Klausur wird aus verschiedenen Fragen bestehen, die zu einem Großteil eindeutig beantwortet werden können (dies erleichtert das korrigieren). Dabei kommen verschiedene Fragestellungen in Betracht, so kann zum Beispiel nach dem Ergebnis eines Befehls gefragt werden. Ebenfalls können Multiple-Choice-Fragen zugrunde gelegt werden. Die Klausur wird wahrscheinlich online durchgeführt, wobei ich mich diesbezüglich noch genau informieren muss, ob das im Rahmen der Prüfungsordnung erlaubt ist. Alternativ wird die Klausur traditionell mit Zettel und Stift durchgeführt.

1.1.6 “Wie soll ich mich auf die Klausur vorbereiten?”

Ich werde allen Klausurteilnehmern die Möglichkeit geben, sich mit möglichen Übungsaufgaben vertraut zu machen.

1.1.7 Seminarwebsite

Die Seminarwebseite zu diesem Seminar finden Sie unter <http://www.lingulist.de/pyjs/>. Dort werden verschiedene Materialien zur Verfügung gestellt, und auch die Slides und Programmierbeispiele für jede einzelne Sitzung können abgerufen werden. Zuweilen gibt es geschützte Inhalte, für deren Abruf ein Passwort benötigt wird. Dieses Passwort wird im Verlaufe des Seminars, und zwar genau JETZT bekannt gegeben.

1.2 Vorstellung

1.2.1 ... meiner Person

Werdegang:

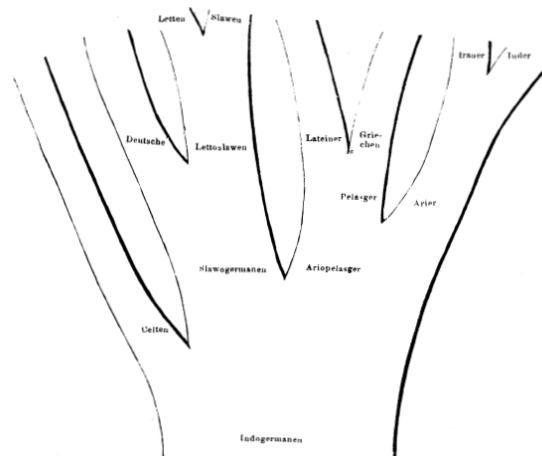
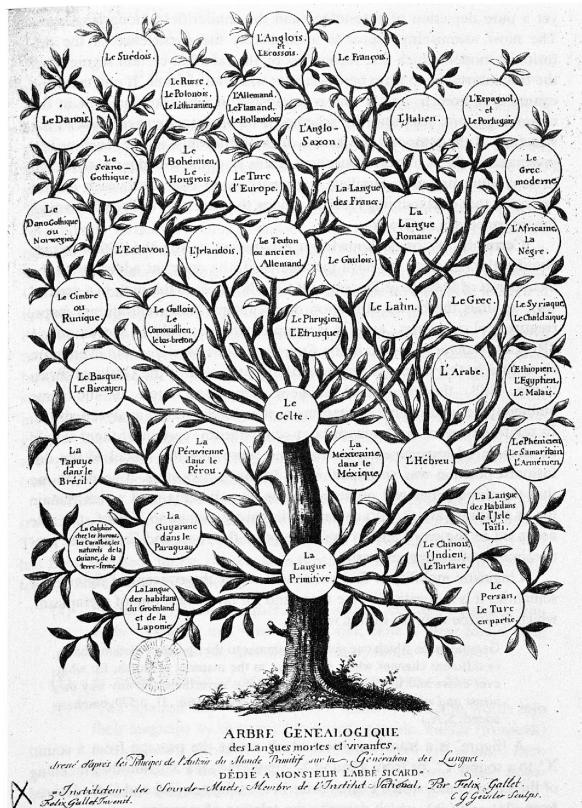
- Jahrgang 1981
- 2002-2008: Studium der Indogermanistik, Sinologie und Slavistik in Berlin
- 2009-2012: Doktorstudium der allgemeinen Sprachwissenschaft in Düsseldorf
- 2012-2014: Post-Doktorand (computergestützter Sprachvergleich) in Marburg
- 2014: Doktorarbeit veröffentlicht als: Sequence Comparison in Historical Linguistics (Düsseldorf University Press)
- 2015-jetzt: DFG Stipendiat (“Computergestützte Untersuchung der chinesischen Sprachgeschichte”)

Wissenschaftliche Interessen:

- historische Linguistik (Sprachwandel, Sprachvergleich, chinesische Dialektologie)
- computerbasierte Ansätze in der historischen Linguistik
- computergestützte Ansätze in der historischen Linguistik

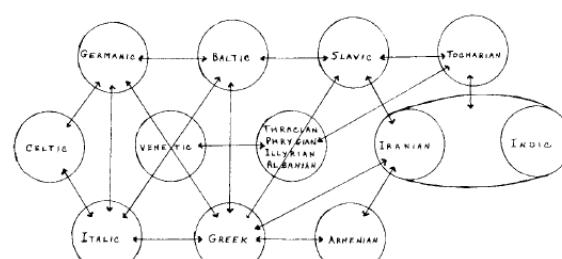
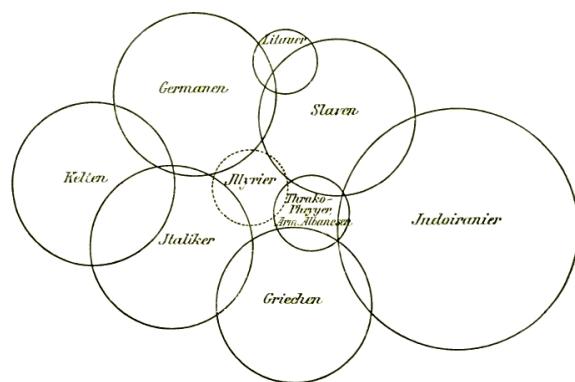
1.2.2 ... des Seminarthemas

Die Entdeckung des Sprachwandels



1.2.3 ... des Seminarthemas

Die Problematik des Sprachwandels



Die Wiederentdeckung der Bäume

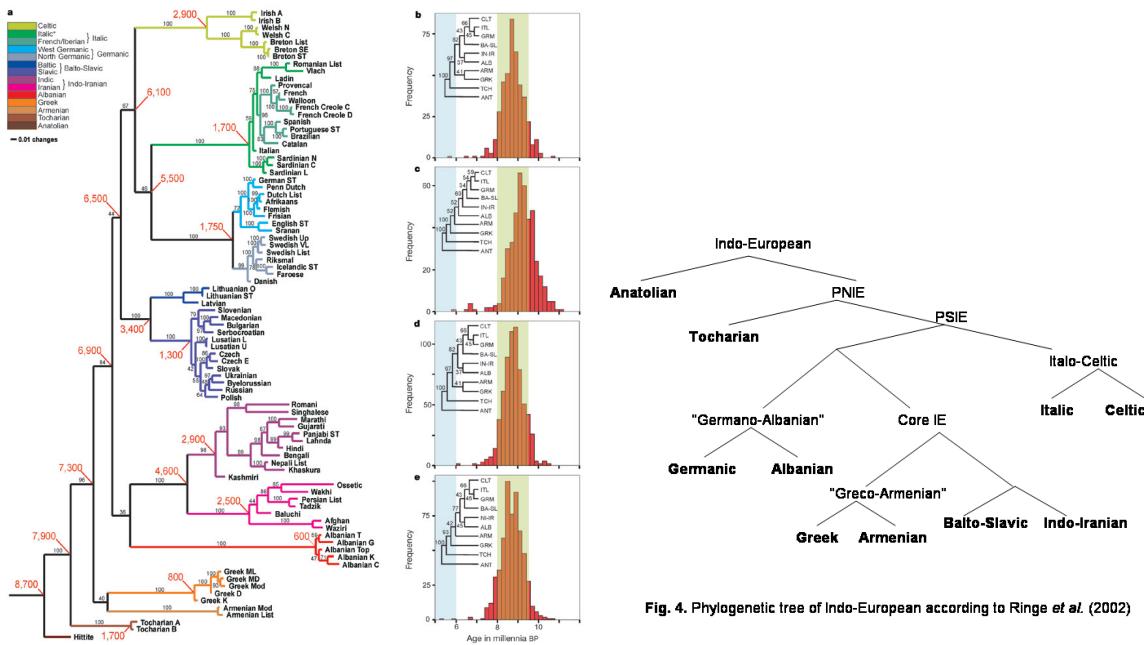
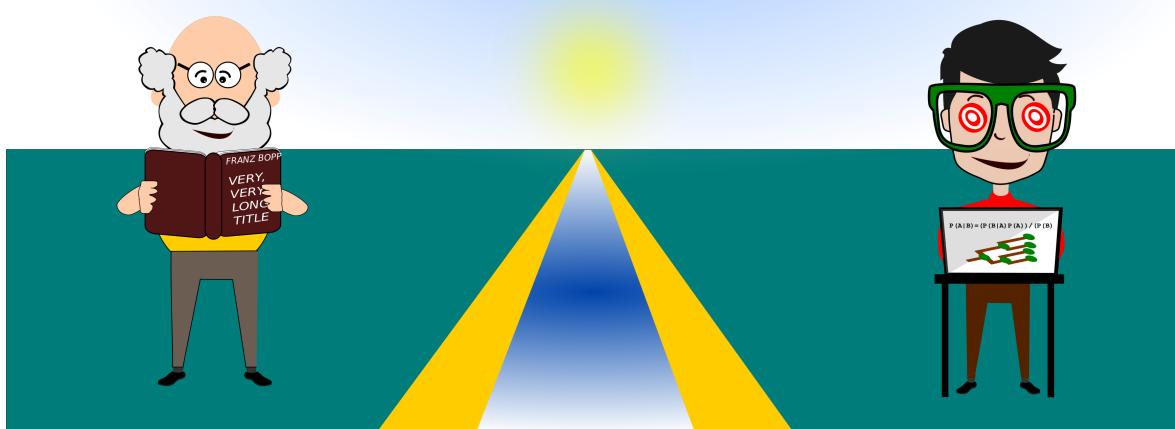
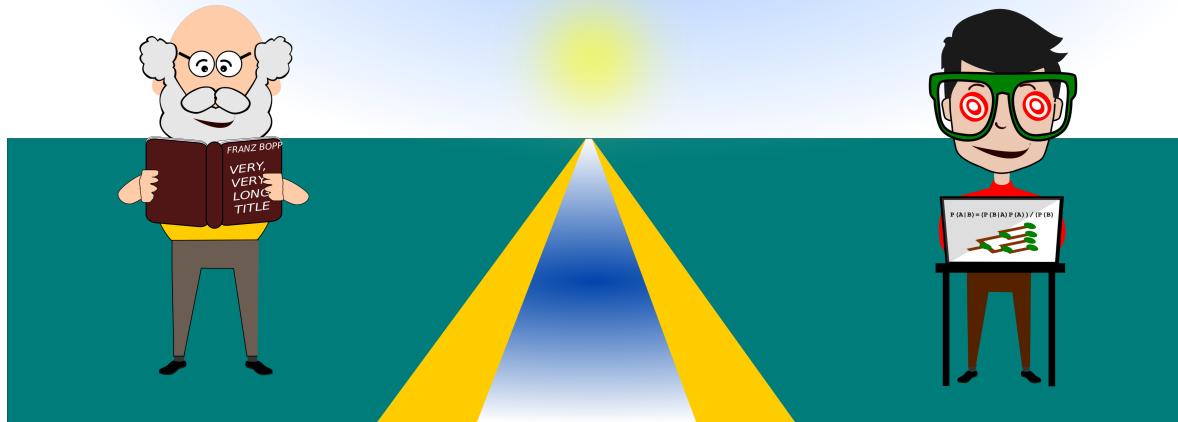


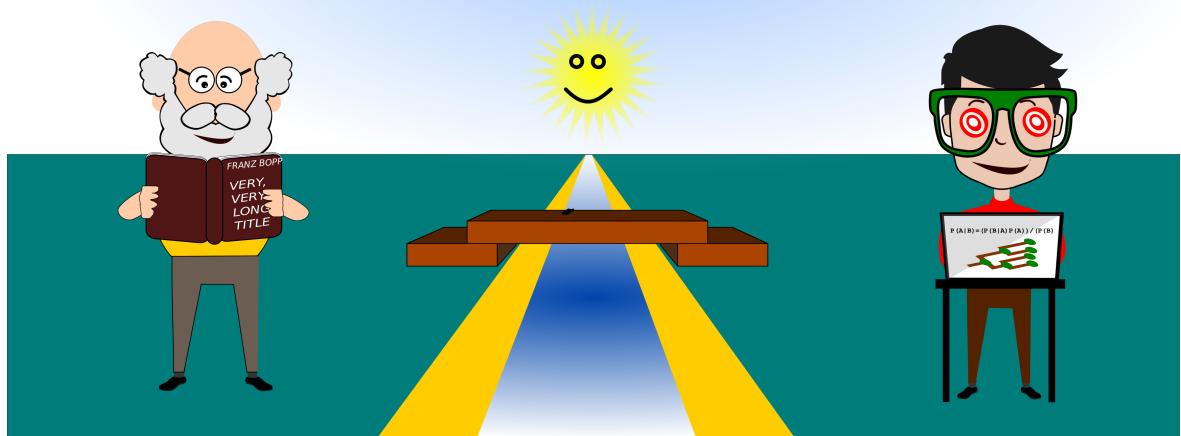
Fig. 4. Phylogenetic tree of Indo-European according to Ringe *et al.* (2002)



PRO: <ul style="list-style-type: none"> - intuition - background knowledge - can juggle with multiple types of evidence 	CONTRA: <ul style="list-style-type: none"> - no intuition - no background knowledge - can't juggle with multiple types of evidence
CONTRA: <ul style="list-style-type: none"> - has to sleep and rest - does not like to count and do boring work - can oversee facts when doing boring work 	PRO: <ul style="list-style-type: none"> - doesn't need to sleep - is very good at counting and boring work - doesn't make errors in boring work



PRO: <ul style="list-style-type: none"> - intuition - background knowledge - can juggle with multiple types of evidence 	CONTRA: <ul style="list-style-type: none"> - no intuition - no background knowledge - can't juggle with multiple types of evidence
CONTRA: <ul style="list-style-type: none"> - has to sleep and rest - does not like to count and do boring work - can oversee facts when doing boring work 	PRO: <ul style="list-style-type: none"> - doesn't need to sleep - is very good at counting and boring work - doesn't make errors in boring work



PRO:

- intuition
- background knowledge
- can juggle with multiple types of evidence

CONTRA:

- has to sleep and rest
- does not like to count and do boring work
- can oversee facts when doing boring work

CONTRA:

- no intuition
- no background knowledge
- can't juggle with multiple types of evidence

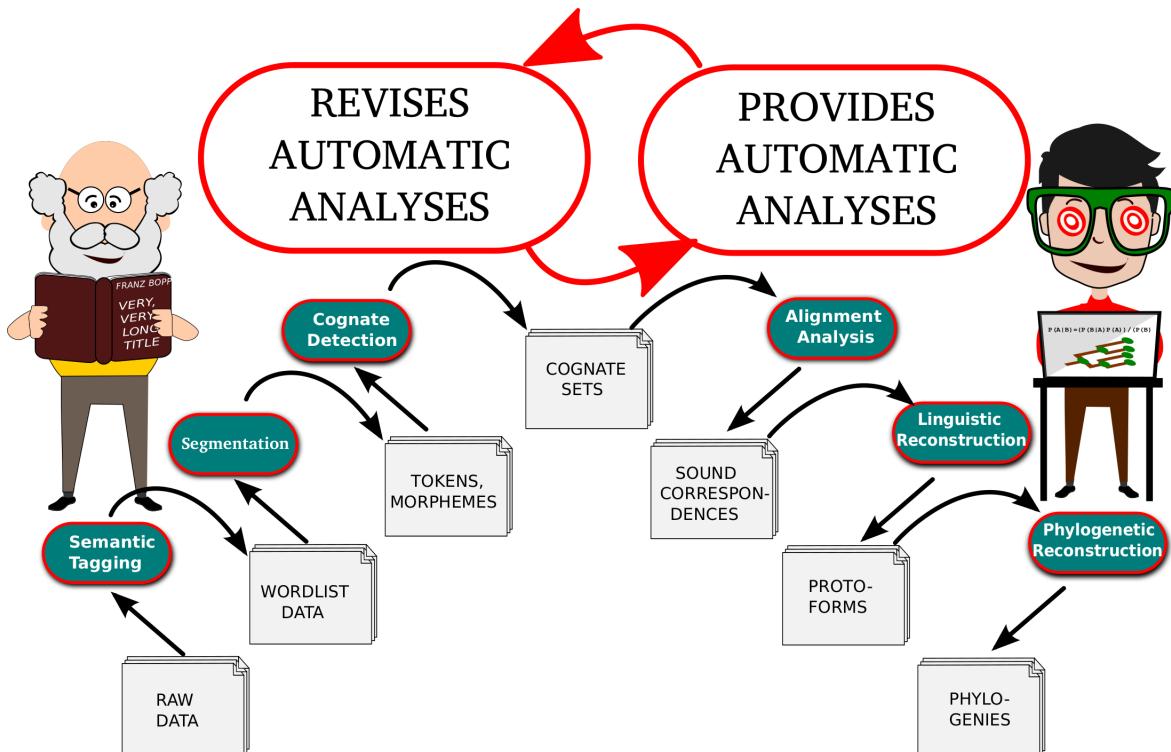
PRO:

- doesn't need to sleep
- is very good at counting and boring work
- doesn't make errors in boring work



COMPUTER-ASSISTED LANGUAGE COMPARISON

Computergestützter Sprachvergleich



1.3 Allgemeines

1.3.1 ... zum Programmieren

Warum ist es sinnvoll, programmieren zu können?

Programmieren zu können ist immer dann sinnvoll, wenn man in seinem Beruf oder den Studien, denen man nachgeht, häufig wiederholte, redundante Operationen ausführen muss, die sich ebensogut automatisch erledigen lassen würden.

Warum ist es sinnvoll, programmieren zu können?

Wenn man zum Beispiel ein psycholinguistisches Experiment mit 200 Stimuli durchführen will, und wissen möchte, wie häufig die Wörter im Durchschnitt vorkommen, dann kann man zur Webseite <http://wortschatz.informatik.uni-leipzig.de/> gehen, wo die Worthäufigkeit für eine Vielzahl von Wörtern verzeichnet ist, und jedes der Wörter einzeln in das Suchfenster eingeben, um dessen Häufigkeit zu ermitteln.

Warum ist es sinnvoll, programmieren zu können?

Dies wird dann mindestens zwei Stunden stumpfer Arbeit zur Folge haben, während der man jedes einzelnen Wort kopiert, die Webseite aufruft, die Häufigkeit kopiert und in eine Tabelle einträgt. Spätestens beim dritten Experiment,

das man durchführt, wird man diese Arbeit hassen und sich Hiwis wünschen.

Daher ist es sinnvoll, programmieren zu können!

Alternativ kann man auch einfach programmieren: Die Leipziger Wortschatzprojekt bietet eine Zusatzbibliothek für Python an (<http://pypi.python.org/pypi/libleipzig>), mit deren Hilfe man ganz schnell ein kleines Programm schreiben kann, das einem für eine beliebige Liste von Wörtern in Sekundenschnelle alle Frequenzen (und noch viel mehr Informationen, wenn man will) aus dem Internet herunterlädt, und — wenn man will, auch noch den Durchschnitt und die Standardabweichung aller Frequenzen errechnet.

Daher ist es sinnvoll, programmieren zu können!

Die Eingabe ist dabei denkbar einfach. Um zum Beispiel die absolute Frequenz des Wortes "Python" zu erhalten, muss man auf der Kommandozeile einfach nur den folgenden Befehl eingeben:

```
>>> Frequencies("Python")  
[(Anzahl: '129', Frequenzklasse: 17)]
```

1.3.2 ... zum Programmieren in der Linguistik

Die Linguistik, insbesondere die historische Linguistik, erlebt derzeit einen Paradigmenwechsel. Während intuitives umfangreiches Fachwissen, das Forscher sich über Jahre intensiven Studiums aneignen mussten, bisher eine sehr große Rolle spielte, und formale Aspekte lediglich als Gedankenspielereien präsentiert wurden, treten im Rahmen der Big-Data-Bewegung nun mehr und mehr die empirischen Aspekte der Disziplin in den Vordergrund.

Die großen Datensammlungen und die leichte Zugänglichkeit von Programmiertools machen es zusehends leichter, verschiedenste Forschungsfragen empirisch zu untersuchen und zu überprüfen. Meiner Meinung nach ist dies sehr wichtig, da die traditionelle Linguistik sich viel zu wenig um die Empirie bemüht hat. Es ist jedoch wichtig, sich im Klaren darüber zu sein, dass eine gute empirische Forschung

immer auf den Errungenschaften der traditionellen Linguistik aufbauen

sollte. Idealerweise praktizieren wir Linguistik als computergestützte Forschung, das heißt, anstelle blind irgendwelchen Algorithmen zu vertrauen, sollten wir Computermethoden entwickeln, die helfen, traditionelle Ansätze zu modellieren und hochwertige Datensätze für die empirische Forschung zu erstellen.

1.4 Spezielles

1.4.1 ... zu Algorithmen, Skripten und Programmen

Was ist Programmieren?

Alan Gauld erklärt den Begriff Programmieren wie folgt:

Computer-Programmierung ist die Kunst, dass ein Computer das macht, was du willst.

Wikipedia ist ein bisschen ausführlicher:

Ein Computerprogramm oder kurz Programm ist eine Folge von den Regeln der jeweiligen Programmiersprache genügenden Anweisungen, die auf einem Computer ausgeführt werden können, um damit eine bestimmte Funktionalität zur Verfügung zu stellen.

Was ist Programmieren?

Etymologisch gesehen, bedeutet Programmieren so viel wie "Vorschriften erstellen" und ist im Deutschen laut Kluge und Seibold (2002) zum ersten Mal seit dem 18. Jahrhunder bezeugt.

Entscheidend für das Programmieren, ist, was die Vorschriften betrifft, dass diese ganz genau befolgt werden, denn so ergibt sich die Möglichkeit, egal, ob das Programm nun von einem Menschen oder einem Computer ausgeführt wird, dass das gewünschte Ergebnis immer erzielt wird.

Was ist ein Algorithmus?

Im Kluge finden wir die Folgende Definition:

Algorithmus. Substantiv Maskulinum, "Berechnungsverfahren", peripherer Wortschatz, fachsprachlich (13. Jh., Form 16. Jh.), mhd. algorismus. Onomastische Bildung. Entlehnt aus ml. algorismus, das das Rechnen im dekadischen Zahlensystem und dann die Grundrechenarten bezeichnet. Das Wort geht zurück auf den Beinamen Al-Hwārizmī ("der Chwā-resmier", eine Herkunftsbezeichnung) eines arabischen Mathematikers des 9. Jhs., durch dessen Lehrbuch die (indischen und dann) arabischen Ziffern in Europa allgemein bekannt wurden. Das Original des hier in Frage kommendes Buches ist verschollen, die ml. Über- setzung ist Liber algorismi de practica arithmeticæ. Die Schreibung mit
in Anlehnung an gr. arithmós "Zahl". [...]

Was ist ein Algorithmus?

Brassard und Bratley (1993) sind da weniger etymologisch:

Das Concise Oxford Dictionary definiert einen Algorithmus als "Verfahren oder Regeln für (speziell maschinelle) Berechnung". Die Ausführung eines Algorithmus darf weder subjektive Entscheidungen

beinhalten noch unsere Intuition und Kreativität fordern. Wenn wir über Algorithmen sprechen, denken wir meistens an Computer. Nichtsdestoweniger können andere systematische Methoden zur Lösung von Aufgaben eingeschlossen werden. So sind zum Beispiel die Methoden der Multiplikation und Division ganzer Zahlen, [...] ebenfalls Algorithmen. [...] Es ist sogar möglich, bestimmte Kochrezepte als Algorithmen aufzufassen, vorausgesetzt, sie enthalten keine Anweisungen wie "nach Geschmack salzen".

- Ein Algorithmus ist eine geordnete Sammlung von Verfahren, mit deren Hilfe eine Aufgabe (ein Problem) eindeutig gelöst werden kann.
- Ein Programm ist eine *Implementierung* von Algorithmen mit Hilfe einer speziellen Programmiersprache.
- Ein Skript ist ein Programm, das in einer interpretierten Programmiersprache (einer Skriptsprache) geschrieben wurde. Alle Programme, die mit Python oder JavaScript erstellt werden, sind demnach Skripte.

1.4.2 ... zur Grundausstattung fürs Programmieren

Texteditoren

Um Skripte zu schreiben, benötigen wir einen guten Texteditor. Das ist nicht das gleiche wie Word oder LibreOffice, sondern ein Editor, der reinen Text schreibt. Aus Zeitgründen erwarte ich von allen Teilnehmern des Seminars, dass sie sich eigenständig einen guten Texteditor zulegen, um Skripte zu schreiben. Ferner ist zu beachten, dass alle Dateien in UTF-8 abgespeichert werden solltet.

Ich selbst schreibe meine Skripte alle mit VIM. Weitere populäre Texteditoren sind:

- GNU Emacs: der natürliche Feind von allen, die gerne VIM benutzen
- Notepad++: ein relativ ordentlicher Texteditor für Windows-Benutzer

Versionsverwaltungssoftware

Wer größere Projekte schreibt, kommt ohne sie nicht aus: die Software zur Versionsverwaltung. Ich selbst verwende Git für meine Arbeit, da es sich wunderbar mit GitHub integrieren lässt, und Daten dadurch auch anderen Nutzern zur Verfügung gestellt werden können. Für das Seminar setze ich voraus, dass jeder Teilnehmer sich grundlegend mit den Ideen hinter Git auseinandersetzt. Um an bestimmte Ressourcen zu gelangen, die ich anbiete, wird ferner ein GitHub Account benötigt werden. Ich empfehle ohnehin allen Programmierinteressierten, sich einen GitHub Account anzulegen, da sich GitHub mehr und mehr zum Standard für kollaboratives Arbeiten entwickelt (was nicht heißt, dass ich es gut finde, dass dahinter ein Konzern steckt!).

Möglichkeiten zum Datenhosting

Wer als Linguist programmiert möchte irgendwann auch seinen Sourcecode anderen Nutzern zur Verfügung stellen. Das ist sehr leicht möglich mit Hilfe

neuer gemeinnütziger Anbieter. Ich empfehle in diesem Zusammenhang Zenodo. Man kann sich mit seinem GitHub Account anmelden, und seinen Code entweder automatisch von Zenodo hosten lassen, oder ihn direkt manuell hochladen. Zenodo garantiert Langzeitarchivierung, ist kostenlos (weil gemeinnützig), und erlaubt bis zu zwei Gigabyte an Daten pro Projekt. Außerdem bekommt man von Zenodo immer automatisch einen Digital Object Identifier, was gewährleistet, dass die Daten im Netz auffindbar und unmissverständlich referenzierbar sind.

1.4.3 ... zu Python und JavaScript

Vorzüge von Python (frei nach Bassi (2010:10))

- **Readability:** Python is a “human-readable language”
- **Built-in features:** Python comes with “batteries included”
- **Availability of third-party modules:** plotting, game development, databases, etc. to model real-world data
- **multi-paradigm:** can be used as a procedural and object-oriented programming language
- **extensibility:** Python can be connected to many other languages
- **open source:** liberal open source license, also for commercial use
- **cross-platform:** works on any computer (even Windows)
- **thriving community:** ask whatever question on stackoverflow, you’ll get an answer in most of the cases

Vorzüge von JavaScript (frei nach Meinung von mir)

- **cross-platform:** JavaScript ist die einzige wirkliche Cross-Platformsprache, weil jeder, der einen Webbrower hat, sie auch hat
- **schön anzusehen:** JavaScript ist sehr hilfreich, wenn man Programme schreiben will, die optisch ansprechend sind
- **leicht zu verwenden (für den Anwender):** JavaScript macht es dem Anwender leicht (dem Programmierer aber leider eher schwer)
- **Schönheit ist nicht alles:** JavaScript ist eine durchweg hässliche Sprache. Dennoch kann man sich mit ihr ganz ordentlich arrangieren.
- **große Unterstützergemeinschaft:** wahrscheinlich sogar größer als die von Python: man findet sehr schnell Antworten zu Fragen im Web
- **viele Third-Party-Modules:** es gibt eine Unmenge von Modulen, auf die man zurückgreifen kann, und noch viel mehr kurze Beispiele

Mit Python und JavaScript hat man zwei unheimlich Mächtige Tools zur Hand, die es einem erlauben, Daten nicht nur auf hochwertige Weise zu analysieren, sondern die Ergebnisse auch noch interaktiv zu präsentieren. Für die moderne, computergestützte Wissenschaft, ist es ein großer Vorteil, über Grundwissen in beiden Sprachen zu verfügen.

2 Erste Schritte in Python

2.1 Allgemeines zu Python

2.1.1 Herkunft

Python

- wurde Anfang der 1990er Jahre von Guido van Rossum entwickelt
- erhielt seinen Namen in Anspielung auf die Monty Pythons
- erschien 1994 in der ersten Version 1.0
- erschien 2000 in der Version 2.0, die viele Neuerungen enthielt
- erschien 2008 in der Version 3.0, die nicht mehr kompatibel mit 2.0 ist, eine Vielzahl von Verbesserungen aufweist und von uns verwendet wird

2.1.2 Charakteristik

Python

- ist eine universelle, interpretierte höhere Programmiersprache
- ist sehr stark auf gute Lesbarkeit ausgerichtet
- relativ einfach zu erlernen
- sehr flexibel (unterstützt objektorientierte und funktionale Programmierung)
- bietet in der Version 3 endlich eine volle Unicode-Unterstützung
- besticht durch einfache Syntax und relativ wenige Schlüsselwörter
- ist eine wunderschöne Sprache

2.1.3 Installation

- Download unter <http://python.org> (wir verwenden ausschließlich Python 3 im Rahmen des Seminars!)
- Installationsanleitung unter <https://www.python.org/about/gettingstarted/>
- die Installation sollte generell problemlos auf fast allen Plattformen verlaufen

2.2 Bibliotheken und Entwickertools

2.2.1 Allgemeines

Wikipedia zur "Programmbibliothek":

Eine Programmbibliothek bezeichnet in der Programmierung eine Sammlung von Programmfunctionen für zusammengehörende Aufgaben. Bibliotheken sind im Unterschied zu Programmen keine eigenständig laufähigen Einheiten, sondern Hilfsmodule, die Programmen zur Verfügung gestellt werden.

Installation von Python-Bibliotheken mit pip

Die Installation von Python-Bibliotheken ist relativ einfach, wenn man auf das Installationstool *pip* ("pip installs packages") zurückgreift. Dieses ist bei den

neuesten Pythonversionen (3.4) bereits vorinstalliert, und kann daher direkt verwendet werden. Die Verwendung ist denkbar einfach. In der Kommandozeile (“cmd” in der Suchleiste bei Windows eingeben) gibt man einfach Folgendes ein:

```
$ pip install packagename
```

Daraufhin wird die Bibliothek dann installiert.

2.2.2 Empfohlene Python-Bibliotheken

- NumPy, “Numeric Python”, <http://numpy.org>: Sehr gute Bibliothek, die eine Vielzahl effizienter numerischer Berechnungen erlaubt und häufig als Grundlage für weitere Bibliotheken gefordert wird.
- SciPy, “Scientific Python”, <http://scipy.org>: Sehr wichtige (leider etwas langsame) Bibliothek für wissenschaftliche Programmierung.
- Matplotlib, <http://matplotlib.org>: Exzellente Bibliothek für das Erstellen hochwertiger Plots von Daten.
- Networkx, <http://networkx.org>: Sehr gute Bibliothek für Netzwerkkalkulationen.
- LingPy, “Linguistic Python”, <http://lingpy.org>: Bibliothek zur Durchführung von quantitativen Analysen in der historischen Linguistik (Download unter: <https://github.com/lingpy/lingpy/releases/tag/v2.4.1-alpha>).

2.2.3 Empfohlene Entwickertools

- IPython, <http://ipython.org>: Sehr gutes Kommandozeileninterpreter für Python, der vor allem auch das Testen von Code ungemein erleichtert.
- Installation unter Windows (kann etwas kompliziert werden, aber Sie sollten das schon schaffen!): <http://ipython.org/ipython-doc/2/install/install.html#windows>
- Installation unter Linux/Ubuntu:

```
$ sudo apt-get install ipython3
```

- Installation unter Archlinux:

```
$ sudo pacman -S ipython
```

2.3 Ein erstes Programmierbeispiel

2.3.1 Das Problem

Karlheinz, ein Düsseldorfer Student der Linguistik im 20. Semester, hat auf einer Mediziner-Party in Köln eine Medizinstudentin kennengelernt, die ihr Physikum bereits abgeschlossen hat, und möchte sie gerne wiedersehen. Dummerweise kann er sich jedoch nicht mehr genau daran erinnern wie sie heißt. Ihr Nachname klang irgendwie nach [maɪə], und ihr Vorname war irgendwas in Richtung [kr̩ɪst̩i:nə], jedoch weiß er nicht, welche Schreibweise er zugrunde legen soll.

Zufällig hat er eine Liste mit den wirklichen Namen und den dazugehörigen Facebook-Namen von allen Bürgern aus Köln und Umgebung als Excel-Tabelle auf seinem Computer zu Hause. Wenn er jetzt noch ein Verfahren finden könnte, das ihm alle Namen anzeigt, die wie [kr̥ist̥i:nə maie] klingen, dann wäre es sicherlich ein Leichtes, herauszufinden, wo die unbekannte Studentin ihre Tierversuche veranstaltet, und sie mit einem Pausenkaffee von . Starbucks zwischen Frosch und Kaninchen zu überraschen...

2.3.2 Der Algorithmus

Die Lösung besteht darin, alle Namen, die auf der Liste auftauchen, in ein anderes Format umzuwandeln, welches den Sprachklang der Wörter wiedergibt und nicht ihre Schreibung. Ein Algorithmus, der für diesen Zweck geschaffen wurde, ist die sogenannte "Kölner Phonetik" (vgl. Postel 1969). Die Kölner Phonetik wandelt Wörter der deutschen Sprache in einen phonetischen Kode um, mit dessen Hilfe auch Wörter, die unterschiedlich geschrieben werden, aber gleich klingen, verglichen werden können. So werden die beiden Namen *Christina Maier* und *Kirsten Mayr* durch den Algorithmus jeweils in die gleiche Zahlenfolge "47826-67" umgewandelt.

Buchstabe	Kontext	Ziffer
A, E, I, J, O, U, Y		0
H		-
B		
P	nicht vor H	1
D, T	nicht vor C, S, Z	2
F, V, W		
P	vor H	3
G, K, Q		
C	im Anlaut vor A, H, K, L, O, Q, R, U, X vor A, H, K, O, Q, U, X außer nach S, Z	4
X	nicht nach C, K, Q	48
L		5
M, N		6
R		7
S, Z		
C	nach S, Z im Anlaut außer vor A, H, K, L, O, Q, R, U, X nicht vor A, H, K, O, Q, U, X	8
D, T	vor C, S, Z	
X	nach C, K, Q	

Das Verfahren

1. Wandle jeden Buchstaben schrittweise um und beachte die Kontextregeln.
2. Reduziere alle mehrfach hintereinander auftauchenden Ziffern auf eine.

3. Entfernen die Ziffer "0" an allen Stellen des Wortes, außer am Anfang.

Aufgabe an alle Seminarteilnehmer

Wenden Sie die Kölner Phonetik auf Ihren Vor- und Nachnamen an und dokumentieren Sie dabei explizit, welche Schritte Sie dabei durchführen. Warum ist das Verfahren komplizierter, als man am Anfang vermuten könnte?

2.3.3 Die Python-Implementierung

Die Kölner Phonetik ist in Python von Robert Schindler implementiert worden und unter der URL (<http://pypi.python.org/pypi/kph/0.3>) erhältlich. Wir ignorieren vorerst die Details der Umsetzung und konzentrieren uns auf die Anwendung. Die Installation ist dank "pip" denkbar einfach:

```
$ pip install kph
```

Die Verwendung auch:

```
>>> import kph
>>> kph.encode("Mattis List")
'628582'
```

Aber wie können wir eine lange Liste von Namen abfragen, ohne jeden Namen einzeln in der Konsole eingeben zu müssen?

Richtig, wir brauchen ein Skript!

Aufbau von Python-Skripten: Shebang und Kodierung

Die erste Zeile von Python-Skripten sieht oft wie folgt aus:

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
```

YOUR CODE HERE

Die erste Zeile, die sogenannte Shebang line, erlaubt es, das Programm durch Doppelklick in Linux- und Mac-Systemen ("unixoide Systeme") auszuführen. Die zweite Zeile legt die Kodierung fest und erlaubt es, bei der Verwendung von Python-2 Programmen, alle Arten von nicht-ASCII Zeichen zu verwenden. Streng genommen brauchen wir aber keine der Funktionen, da wir Programme auf Unix-Computern auch leicht über die Konsole ausführen können, und Python3 UTF-8 voll unterstützt.

Aufbau von Python-Skripten: Kommentare

```
# Dies ist eine Kommentarzeile,
# durch welche es möglich ist, Dinge in das
# Programm zu schreiben, die nachher
```

```

# nicht ausgeführt werden.

# Man kann Kommentare Überall einsetzen, man
# muss aber beachten, dass, wenn man
# sie einsetzt, alles was hinter dem Kommentar-Zeichen folgt, nicht mehr
# interpretiert wird.

1 + 1 # wird hier interpretiert
# 1 + 1 wird nicht interpretiert
1 + # ruft einen Fehler hervor, weil das "+" ohne Gegenstück ist

"Alternativ kann man Dinge auch in Anführungsstriche setzen.

"""

Sicherer ist es allerdings, drei Anführungsstriche auf
einmal zu verwenden, weil man dann auch Anführungsstriche
innerhalb des Kommentars benutzen kann, bzw. auch über
mehrere Zeilen hinweg schreiben kann.

"""

```

Die Print-Funktion

```

# file: hallo_welt.py
print("Hallo Welt!")

$ python3 hallo_welt.py
Hallo Welt!

```

Einbinden von Modulen (Bibliotheken)

```

# file: test_kph.py

import kph
# importiert die Kölner Phonetik, vorher kann man
# die Bibliothek nicht verwenden

print(kph.encode('Monty Python'))

$ python3 test_kph.py
662126

```

Einige kurze Ideen zum Experimentieren und Nachdenken

1. Experimentieren Sie mit der Kölner Phonetik. Welche Ausgaben erhalten Sie, wenn Sie

- eine Zahl in Anführungsstrichen,
 - eine Zahl ohne Anführungsstriche, oder
 - Buchstaben mit diakritischen Zeichen ("áłń") eingeben?
2. Gibt man statt "Kirsten", "Maier" die Zeile `encode("Kirsten Maier")` ein, verändert sich die Ausgabe. Woran liegt das?
 3. Was brauchen wir (welche Routinen, Verfahren), um alle Daten aus der Excel-Tabelle von Karlheinz einlesen und in ihre Werte entsprechend der Kölner Phonetik umwandeln zu können?

3 Erste Schritte in JavaScript

3.1 Allgemeines zu JavaScript

3.1.1 Herkunft

JavaScript

- ist eine Skriptsprache, die speziell für dynamisches HTML in Webbrowsern entwickelt wurde
- verändert Inhalte von Webseiten, reagiert auf Benutzereingaben, und erweitert die Möglichkeiten von HTML und CSS
- ist ein abtrünniges Kind von C, was die hässliche Syntax angeht und hat mit Java selbst sehr wenig zu tun (obwohl die Syntax von Java auch sehr hässlich ist)
- wurde ursprünglich von Netscape entwickelt und im Jahre 1996 in der Version 1.1 veröffentlicht

3.1.2 Charakteristik

JavaScript

- hat eine sehr hässliche Syntax
- ist prinzipiell nicht sehr schwer zu erlernen, sieht aber hässlich aus
- erlaubt es den Benutzern, sehr, sehr hässlichen Kode zu schreiben
- ist am Anfang sehr gewöhnungsbedürftig, bis man begriffen hat, wie die Interaktion zwischen HTML, CSS, und JavaScript abläuft
- ist generell "innen pfui, außen hui"
- macht des dem Benutzer sehr schwer, modularen Kode zu schreiben

3.1.3 Installation

JavaScript ist fester Bestandteil gängiger Webbrowser, wie Firefox, Chrome, oder Safari. Im Internet-Explorer ist JavaScript angeblich auch vorinstalliert, jedoch laufe viele Apps sehr viel schlechter als gewünscht, weil Microsoft sich mal wieder qualitativ von den anderen Anbietern abgrenzen muss.

3.2 Bibliotheken und Entwickertools

3.2.1 Webbrowser

Ich empfehle, beim Entwickeln von JavaScript-Applikationen grundsätzlich auf einen der gängigen Unix-Browser (Firefox, Chrome, oder Safari) zurückzugreifen.

Firefox und Chrome, welche ich beide regelmäßig verwende, unterscheiden sich in den Funktionen, die sie bieten. Firefox erlaubt bestimmte Dateizugriffe, ohne die Applikation über einen Server laufen zu lassen, was die Entwicklung von Applikationen erleichtert. Chrome hat Vorteile im Layout und im Debugging. Grundsätzlich sollten alle Applikationen mindestens auf Firefox und Chrome getestet werden, da es hier mitunter zu bestimmten Unterschieden kommen kann. Auch unerwartete Fehler können in einem Webbrowser auftauchen, im anderen aber nicht.

3.2.2 Bibliotheken

Empfohlene Frameworks:

- jQuery: eine der am weitesten verbreiteten JavaScript-Bibliotheken, die viele Erweiterungen und Erleichterungen bietet und als Grundlage zahlreicher Plugins dient. Aufgrund der Größe von jQuery sollte man sich jedoch für jedes Projekt überlegen, ob man die Bibliothek auch wirklich braucht, denn auch JavaScript allein bietet viele Möglichkeiten, kurzen und knackigen Kode zu schreiben.
- d3: Eine wunderbare Bibliothek zur Datenvisualisierung, ein bisschen gewöhnungsbedürftig in der Anwendung, aber unheimlich schön in den Ergebnissen. Entwickelt wurde die Bibliothek vorrangig von Mike Bostock, der für die New York Times arbeitet, die seine Pionierarbeit der interaktiven Visualisierung sponsort.

3.2.3 Entwickertools

Man kann JavaScript auch in der Konsole ausführen (die selbst in JavaScript geschrieben wurde):

Zum Testen bietet sich node.js an: Dieses Paket erlaubt es, JavaScript Kode in Skripten auszuführen und bietet auch eine interaktive Konsole. Der Vorteil von node.js ist, dass man Skripte testen kann, ohne den ansonsten erforderlichen HTML/CSS-Überbau zugrunde legen zu müssen.

3.3 Ein erstes Programmierbeispiel

3.3.1 Die Kölner Phonetik in JavaScript

Es existiert natürlich bereits eine Implementierung der Kölner Phonetik für JavaScript, implementiert von J. Tillmann: <https://github.com/jtillmann/colophoneticjs>. Diese Version wurde für die Zwecke unseres Seminars leicht umgeschrieben, so dass wir sie in ähnlicher Weise, wie die Python-Version der Kölner Phonetik verwenden können. Der resultierende Source Code, ein Skript mit Namen kph.js befindet sich auf der Projektwebseite und kann dort heruntergeladen werden. Er wurde ebenfalls bereits in die Terminal-Applikation integriert.

Kölner Phonetik mit Hilfe des Web-Terminals:

```
js> kph.encode('Mayer');
67
```

Kölner Phonetik mit node.js:

Einbinden von kph.js:

```
> kph = require('./js/kph.js') /* Im Order demos/ */
{ encode: [Function] }
```

Anwenden:

```
> kph.encode('Müller-Lüdenscheidt');
'65752682'
> kph.encode('Muller-Ludenscheidt Sebastian Bach')
65752682 81826 14
```

Die Verwendung in der Konsole ist aber noch nicht alles! Viel interessanter wird es ja, wenn man den Kode gleich auf einer Webseite einsetzen kann, zum Beispiel als interaktive Applikation.

Dafür brauchen wir zunächst eine HTML-Eingabe und Ausgabe, die wir in einer HTML-Seite (mit Standard-Head-Body Struktur) wie folgt unterbringen können:

```
OK</>
```

Die Ausgabe sieht dann auf der Webseite wie folgt aus:

OK

Um eine bestimmte JavaScript-Bibliothek ausführen zu können, müssen wir sie im Header der HTML-Datei (oder an einer beliebigen anderen Stelle) einbinden, was wie folgt aussieht:

```
<html>
  <head>

  </head>
  <body>

    GIB MIR DIE KPH!

  </body>
</html>
```

Dann brauchen wir noch eine JavaScript Funktion, die mit HTML kommuniziert:

```

// Funktion greift auf die HTML-Datei zu und Berechnet
// die Werte für die Kölner Phonetik
function showKPH() {
    /* get the input button element */
    var ipt = document.getElementById('ipt');
    /* get the input value */
    var ipt_val = ipt.value;
    /* convert to Kölner Phonetik */
    var converted_values = kph.encode(ipt_val);
    /* write to html page */
    var opt = document.getElementById('opt');
    opt.innerHTML = converted_values;
}

```

Beachten Sie, dass es zwei Arten in JS gibt, um Kommentare einzufügen, den Doppelslash (//), der den Rest einer Zeile auskommentiert, und die Kombination von Slash mit Asterisk (* und *), mit denen man auch über Zeilen hinaus auskommentieren kann.

Diese Datei können wir entweder als separates Skript abspeichern und in HTML einbinden,

```

<html>
    ...
</body>

```

oder wir können sie direkt zwischen die Skript-Tags schreiben

```

<html>
    ...
function showKPH() {
    var ipt = document.getElementById('ipt');
    var ipt_val = ipt.value;
    var converted_values = kph.encode(ipt_val);
    var opt = document.getElementById('opt');
    opt.innerHTML = converted_values;
}
</body>

```

Und so sieht das Ganze dann in Aktion aus:

4 Datentypen und Variablen

4.1 Variablen

4.1.1 ... im Allgemeinen

Was ist eine Variable?

- *Variable* ist das Substantiv zu *variabel*, welches auf Latein *variabilis* “veränderbar” zurückgeht.
- Eine Variable ist also etwas, das änderbar ist.
- Johnny Depp ist ein Beispiel für eine *Variable*, weil er sehr änderbar ist, wie man an seinen Filmen sehen kann.

Wikipedias Definition:

In computer programming, a variable is a symbolic name given to some known or unknown quantity or information, for the purpose of allowing the name to be used independently of the information it represents. A variable name in computer source code is usually associated with a data storage location and thus also its contents, and these may change during the course of program execution.

Definition in Puttkamer (1990):

Bei jeder Art von Datenverarbeitung beziehen sich die Anweisungen auf direkte Daten, die eingegeben werden oder auf Variable, die man sich vorstellen kann als mit Namen versehen Behälter für Daten. Jede Zuweisung an eine Variable füllt Daten in diesen Behälter, und der Wert einer Variablen ist der Inhalt dieses Behälters. So ist z. B. der Effekt einer Anweisung $A := 3 + 7$: Bilde mit den direkt eingegebenen Daten 3 und 7 die Summe 10 und weise diesen Wert der Variablen mit Namen A zu. Ein [sic!] anschließende Anweisung “drucke A” druckt den Inhalt des Behälters A aus, den Wert der Variablen A, hier 10.

Vereinfachte Begriffserklärung

- Beim Programmieren werden Werte benötigt, deren Inhalt variabel ist.
- Variabel heißt, dass die Werte sich entweder bei jedem erneuten Programmaufruf ändern können, oder sogar innerhalb des Programms.
- Bspw. lässt sich ein Programm, dass eine Ganzzahl mit sich selbst multipliziert und das Ergebnis dieser Multiplikation wieder mit sich selbst, nicht schreiben, wenn man nicht eine Möglichkeit hat, auf die Zahl zuzugreifen, obwohl man sie noch nicht kennt.
- Variablen ermöglichen derartige Programmoperationen.
- Variablen sind Platzhalter, die, wenn das Programm ausgeführt wird, mit einem Wert gefüllt werden (*Deklaration*).

4.1.2 ... in Python

Deklaration von Variablen in Python

```

>>> VARIABLE = VALUE
>>> VAR1, VAR2, ... = VAL1, VAL2, ...
>>> VAR1, \*VAR2 = VAL1, VAL2, VAL3, ...
>>> VARIABLE = VAL1, VAL2, ...

>>> a = 1
>>> b, c = 2, 3
>>> d, \*e = 4, 5, 6
>>> f = 7, 8, 9
>>> print(myvar, myvar1, myvar2, myvar3, myvar4, myvar5)
1 2 3 4 [5, 6] (7, 8, 9)

```

Struktur von Variablennamen

```

>>> Name = 1
>>> Name_von_mir = 1
>>> Name2 = 1

>>> 2Name = 1
SyntaxError: invalid syntax

```

Programmierbeispiele

```

>>> print("Variable")
Variable
>>> Variable = "Variable"
>>> print(Variable)
Variable
>>> Variable
'Variable'
>>> Variable == Variable
True
Variable == 'Variable'
True
>>> var1, var2 = 2, 3
>>> print(var1,"und", var2,"macht",var1 + var2)
2 und 3 macht 5

```

Fehlermeldungen

```

>>> test = 1,2
>>> test = bla
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'bla' is not defined

```

```

>>> print(test1)
traceback (most recent call last):
  file "<stdin>", line 1, in <module>
nameerror: name 'test1' is not defined

>>> test1 = 2
>>> test2 = "2"
>>> test1 + test2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'

```

4.1.3 ... in JavaScript

Deklaration von Variablen in JavaScript

```

> VARIABLE = 1;
> var VAR = 1;
> VAR1 = 1; VAR2 = 2;
> var VAR1, VAR2;
> VAR1 = 1; VAR2 = 1;

```

Das Schlüsselwort “var” wird in JavaScript verwendet, um sicherzustellen, dass Variablen in Funktionen lokal definiert werden. Grundsätzlich sollte man (da wir ohnehin nicht global definieren sollten) darauf achten, immer das Schlüsselwort “var” einer Variablen-deklaration vorwegzustellen.

Struktur von Variablennamen

```

> var Name = 1
> var Name_von_mir = 1
> var Name2 = 1

> var 2Name = 1
SyntaxError: identifier starts immediately after numeric literal

```

Programmierbeispiele

```

> var myvar = 1;
> var myvar2 = 2;
> var myvar3, myvar4, myvar5;
> myvar3 = 3; myvar4 = 4; myvar5 = 5;
> [myvar1, myvar2, myvar3, myvar4, myvar5].map(function(x){return x+x});
[ 2, 4, 6, 8, 10 ]

```

Fehlermeldungen

```
> var a = 10;  
> var b = '11';  
> a + a;  
20  
> b + b;  
1111  
> a + b  
1011  
> a - "10";  
0  
> a * "10";  
100
```

Vorsicht mit den Operatoren +, -, * und / in JavaScript! Ihr Verhalten kann unberechenbar sein, da im Gegensatz zu Python kein Typcheck durchgeführt wird!

4.2 Datentypen

4.2.1 ... im Allgemeinen

Wikipedia-Definition:

Formal bezeichnet ein Datentyp in der Informatik die Zusammenfassung von Objektmengen mit den darauf definierten Operationen. Dabei werden durch den Datentyp des Datensatzes unter Verwendung einer so genannten Signatur ausschließlich die Namen dieser Objekt- und Operationsmengen spezifiziert. Ein so spezifizierter Datentyp besitzt noch keine Semantik. Die weitaus häufiger verwendete, aber speziellere Bedeutung des Begriffs Datentyp stammt aus dem Umfeld der Programmiersprachen und bezeichnet die Zusammenfassung konkreter Wertebereiche und darauf definierten Operationen zu einer Einheit. Zur Unterscheidung wird für diese Datentypen in der Literatur auch der Begriff Konkreter Datentyp verwendet. Für eine Diskussion, wie Programmiersprachen mit Datentypen umgehen, siehe Typisierung.

Vereinfachte Begriffserklärung

- Variable als Platzhalter, der mit einem bestimmten Wert gefüllt wird.
- Worum es sich bei dem Wert handelt, ist wichtig für die korrekte Durchführung eines Programms.
- Wörter kann man beispielsweise nicht addieren.
- Zahlen kann man dafür nicht einfach aneinanderreihen.

- Auch im wirklichen Leben teilen wir unsere Zeichen in gewisser Weise in Datentypen ein, denn wenn wir den Satz "Multipliziere mal 1 und 1" hören, dann denken wir bei "1" an eine Zahl und nicht an eine Zeugnisnote, weil man eine Zeugnisnote nicht multiplizieren kann.

4.2.2 ... in Python

Allgemeines

In Python werden Datentypen bei der Variablen Deklaration nicht explizit angegeben, sondern aufgrund der Struktur der Werte, die den Variablen zugewiesen werden, automatisch bestimmt. Python weist eine Vielzahl von Datentypen auf und ermöglicht aufgrund seiner objektorientierten Ausrichtung auch die Erstellung eigener komplexer Datentypen.

Wichtigste Datentypen in Python

- *integer*: fasst ganzzahlige Werte (2, 3, -5, 0)
- *float*: fasst Fließkommawerte (1.2, 1.222, -0.5, -2.5)
- *string*: fasst Zeichenketten ("Friedrich", "Nietzsche", "[]")
- *list*: fasst jede Art von anderen Datentypen in linearer Anordnung und kann verändert werden (["Friedrich", "2", 1998, -0.5])
- *tuple*: fasst jede Art von anderen Datentypen, kann aber nicht verändert werden (("Friedrich", "2", 1998, -0.5))
- *dict*: fasst jede Art von anderen Datentypen als Sammlung von Key-Value-Paaren, wobei der Key weder Liste noch Dictionary sein kann ({1:1, "Friedrich": "Nietzsche"})

Überprüfen

```
>>> a, b, c, d = 1, "2", 2.5, [1,2]
>>> type(a)
<class 'int'>
>>> print(type(b), type(c))
<class 'int'> <class 'str'>
>>> isinstance(d, list)
True
>>> isinstance(d, (int,str))
False
```

Programmbeispiele

```
>>> a, b, c = 1, "2", 3.5
>>> words = ["apfel", "wurst", "gurke"]
```

```

>>> nahrungs_typ = {"apfel": "vegan", "wurst": "carnivor", "gurke": "vegan"}
>>> b + b
'22'
>>> a / c
-2.5
>>> type(a/c)
<class 'float'>
>>> for word in words: print(word, nahrungs_typ[word])
apfel vegan
wurst carnivor
gurke vegan
>>> print(words[1])
wurst

```

4.2.3 ... in JavaScript

Allgemeines

Auch in JavaScript werden Datentypen bei der Deklaration nicht explizit angegeben, sondern dynamisch bestimmt. Auch in JavaScript können komplexe Datentypen aufgrund der Möglichkeit, objekt-orientiert zu programmieren, erstellt werden. Im Gegensatz zu Python ist es viel leichter, Operationen auf unterschiedlichen Datentypen durchzuführen, was problematisch werden kann, da die erwarteten Ergebnisse sich leicht unterscheiden können, wenn ein Programm nicht sorgfältig geprüft wird.

Wichtigste Datentypen in Javascript

- *number*: Ganz- und Fließkommazahlen (1, 1.5)
- *string*: Zeichenketten ("hallo", 'welt')
- *array*: linear angeordnete variable Datentypen ([1, "2", -3])
- *object*: Key-Value-Paare ({0:1, 1:2})

Beachten Sie, dass es sich bei dem Datentyp "array" offiziell um eine spezielle Form des sehr abstrakten Datentypen "object" handelt, weshalb ein Type-Check auch für einen Array immer den Wert "object" zurückliefern wird.

Überprüfen

```

> var a = 1;
> var b = 1.5;
> var c = "1";
> var d = [a, b, c];
> var e = {0 : a, 1 : b, 2 : c};
> typeof a;
number
> typeof b;

```

```
number  
> typeof c;  
string  
> typeof d;  
object  
> typeof e;  
object
```

Programmbeispiele

```
> var a = 1;  
> var b = 1.5;  
> var c = "1";  
> var d = [a, b, c];  
> var e = {0 : a, 1 : b, 2 : c};  
> a == d[0];  
True  
> a === d[0];  
True  
> e[0] = 2  
for (key in e) {alert(key+' '+e[key])}
```

4.3 Praktische Beispiele

4.3.1 Die Dreiballkaskade

Dreiballkaskade in Wikipedia

Als Kaskade wird das am einfachsten zu erlernende Jongliermuster mit einer ungeraden Anzahl von Gegenständen (Zum Beispiel: Bällen, Keulen oder Ringen) bezeichnet. Dabei wird mit zwei Gegenständen in einer Hand und einem in der anderen Hand angefangen. Der erste Wurf wird durch die Hand ausgeführt, in der zwei Gegenstände sind. Wenn der Gegenstand den höchsten Punkt erreicht, wird der Gegenstand aus der anderen Hand losgeworfen (und zwar unter dem zuvor geworfenen Gegenstand hindurch). Dadurch ist diese Hand frei, um den ersten Gegenstand zu fangen. Wenn der zweite Gegenstand am höchsten Punkt angekommen ist, wird der dritte Gegenstand losgeworfen (mit der Hand, die auch den ersten Gegenstand geworfen hat) und so weiter.


```

gegenstand2 = '(2)'
gegenstand3 = '(3)'

# halte fest, welcher gegenstand gerade wo ist
RechteHand = gegenstand2
LinkeHand = gegenstand3
Rechts0ben = gegenstand1
Links0ben = ' '

# Jetzt kann es losgehen. Das Programm startet, indem wir die Entertaste
# drücken.
input("Los geht's!")

# Wir führen eine Schleife aus (Details dazu kommen später)
i = 0
while i < 20:

    # wir deklarieren eine variable snapshot, die in jeweils vier schritten
    # durch das derzeit vorliegende jongliermuster ersetzt wird
    SnapShot = JonglierMuster
    SnapShot = SnapShot.replace('(R)',Rechts0ben)
    SnapShot = SnapShot.replace('(L)',Links0ben)
    SnapShot = SnapShot.replace('(l)',LinkeHand)
    SnapShot = SnapShot.replace('(r)',RechteHand)

    # Wir benutzen nicht print, um das ganze auszugeben, sondern input(),
    # weil damit immer gleichzeitig auch eine Pause verbunden ist (erst wenn
    # man Enter drückt geht es weiter). (in Python2 brauchen wir "raw_input")
    input(SnapShot)

    # jetzt passen wir die variablen an, wobei wir schauen, wo gerade der
    # ball ist.
    if Rechts0ben == ' ':
        Rechts0ben = LinkeHand
        LinkeHand = ' '
        i += 1

    elif RechteHand == ' ':
        RechteHand = Rechts0ben
        Rechts0ben = ' '
        i += 1

    elif LinkeHand == ' ':
        LinkeHand = Links0ben
        Links0ben = ' '
        i += 1

    elif Links0ben == ' ':
        Links0ben = RechteHand

```

```
RechteHand = ' '
i += 1

# Wenn alles geschafft ist, kann man schon mal darauf hinweisen, dass das
# ziemlich anstrengend ist...
input("Puh, war das anstrengend!")
```

Dies ist eine Python-2 Version, das heißt, dass die Funktion "input" durch "raw_input" ersetzt werden muss:

4.3.3 Die Dreiballkaskade in Javascript

HTML Kode

```
<html>
<head>
  Kaskade Demo

</head>
```

(1)

(2)

(3)

Next Catch

...

CSS Kode

```
.white {
  color: red;
  width: 50px;
  height: 50px;
  background: lightgray;
}
```

```

.hand {
    border-bottom: 10px solid Orange;
}
.right {
    border-right: 10px solid Orange;
}
.left {
    border-left: 10px solid Orange;
}

.ball {
    font-size: 20px;
    color: Crimson;
    font-weight: bold;
    text-align: center;
    background: lightgray;
}

```

JavaScript Kode

```

function nextCatch() {

    // get items for all the values in the cells
    var ro = document.getElementById('ro');
    var lo = document.getElementById('lo');
    var ru = document.getElementById('ru');
    var lu = document.getElementById('lu');

    console.log(ro.innerHTML, lo.innerHTML);

    // get empty value
    if (ro.innerHTML == '') {
        ro.innerHTML = lu.innerHTML;
        lu.innerHTML = '';
    }
    else if (lo.innerHTML == '') {
        lo.innerHTML = ru.innerHTML;
        ru.innerHTML = '';
    }
    else if (lu.innerHTML == '') {
        lu.innerHTML = lo.innerHTML;
        lo.innerHTML = '';
    }
    else if (ru.innerHTML == '') {
        ru.innerHTML = ro.innerHTML;
        ro.innerHTML = '';
    }
}

```

DEMO

5 Operatoren und Kontrollstrukturen

5.1 Operatoren

Wikipedia zu Operatoren in der Mathematik

Ein Operator ist eine mathematische Vorschrift (ein Kalkül), durch die man aus mathematischen Objekten neue Objekte bilden kann. Er kann eine standardisierte Funktion oder eine Vorschrift über Funktionen sein. Anwendung finden die Operatoren bei Rechenoperationen, also bei manuellen oder bei maschinellen Berechnungen.

Wikipedia zu Operatoren in der Logik

Ein Logischer Operator ist eine Funktion, die einen Wahrheitswert liefert. Bei der zweiseitigen, booleschen Logik liefert er also wahr oder falsch, bei einer mehrwertigen Logik können auch entsprechend andere Werte geliefert werden.

5.1.1 Allgemeines

Vereinfachte Begriffserklärung

- Wenn wir uns den Operator als einen Verfertiger von Objekten vorstellen, dann heißt das, dass ein Operator *irgendetwas* mit *irgendetwas* anstellt. Entscheidend sind hierbei die Fragen, *was* der Operator tut, und *womit* der Operator etwas tut.
- Ein Operator wandelt also eines oder mehrere Objekte in neue Objekte um.
- Das, was umgewandelt wird, nennen wir die *Operanden* eines Operators.
- Das, was der Operator mit den Operanden tut, nennen wir die *Operation*.
- Der Plus-Operator wandelt bspw. zwei Zahlen in eine Zahl um, indem er die Operation *Addition* ausführt.

Klassifikation von Operatoren

Operatoren können nach verschiedenen Kriterien klassifiziert werden. Die grundlegendste Unterscheidung besteht in der Anzahl der Operanden, auf die ein Operator die Operation anwendet. Hierbei unterscheiden wir zwischen *unären* und *binären* Operatoren

- *Unäre* Operatoren haben nur einen Operanden.

- *Binäre* Operatoren haben zwei Operanden.

Klassifikation von Operatoren

Ferner kann zwischen *arithmetischen*, *logischen* und *relationalen* Operatoren unterschieden werden.

- *Arithmetische* Operatoren führen arithmetische Operationen aus (Addition, Division, etc.).
- *Logische* Operatoren liefern Aussagen über Wahrheitswerte (wahr oder falsch).
- *Relationale* Operatoren liefern Aussagen über die Beziehungen zwischen Objekten (Identität, Zugehörigkeit).

Überladen von Operatoren

- Normalerweise werden Operatoren immer nur für spezifische Datentypen definiert.
- Der Operator `+` nimmt als Operanden bspw. gewöhnlich nur Zahlen.
- In vielen Programmiersprachen ist es jedoch üblich, Operatoren, je nach Datentyp, auf den sie angewendet werden, unterschiedliche Operationen zuzuschreiben.
- Diesen Vorgang nennt man das *Überladen* von Operatoren.
- Wird der Additionsoperator `+` bspw. auf den Datentyp *string* angewendet, so bewirkt er eine Verkettung von Strings (Konkatenation).
- Das Überladen von Operatoren ermöglicht es, sehr kompakt und flexibel zu programmieren.

5.1.2 ... in Python

Allgemeines

- Die Operatoren in Python ähneln denen vieler Programmiersprachen.
- Neben den durch mathematische Zeichen dargestellten Operatoren (`+`, `-`, `*`) sind einige Operatoren auch als *Namen* (`is`, `in`) definiert.
- Die Überladung von Operatoren ist ein entscheidender Wesenszug der Sprache. Viele Operatoren sind auf viele Datentypen anwendbar.

Arithmetische Operatoren

Operator	Klass.	Name	Erläuterung	Beispiel
+	unär	?	verändert den Operanden nicht	+x
-	unär	Negation	verändert den Operanden	-x
+	binär	Addition	liefert die Summe der Operanden	x + y
-	binär	Subtraktion	liefert die Differenz der Operanden	
*	binär	Multiplikation	liefert das Produkt der Operanden	x * y
/	binär	Division	liefert den Quotienten der Operanden als Gleitkommazahl	x / y
//	binär	Division	liefert den Quotienten der Operanden als Ganzzahl	x // y
%	binär	Modulo	liefert den Rest der Division der Operanden	x % y
**	binär	Potenz	liefert die Potenz der Operanden	x ** y

Arithmetische Operatoren

```

>>> x,y = 5,2
>>> +x
5
>>> -x
-5
>>> x + y
7
>>> x - y
3
>>> x * y
10
>>> x / y
2.5
>>> x // y
2
>>> x % y
1
>>> x ** y
25

```

Logische Operatoren

- Die logischen Operatoren dienen dem Verarbeiten von Wahrheitswerten.
- Jeder Wert bekommt in Python automatisch einen Wahrheitswert zugewiesen (**True** oder **False**).
- Die Wahrheitswerte werden dem Datentyp *bool* zugeschrieben.
- Negative Zahlen einschließlich der Zahl 0 besitzen den Wahrheitswert **False**.
- Alle "leeren" Werte (der leere String "" oder die leere Liste []) besitzen ebenfalls den Wahrheitswert **False**.
- Die logischen Operatoren prüfen den Wahrheitswert von Werten und liefern als Ergebnis einen Wahrheitswert zurück.

Logische Operatoren

Operator	Klass.	Name	Erläuterung	Beispiel
not	unär	Negation	kehrt den Wahrheitswert des Operanden um	not x
and	binär	Konjunktion	verknüpft die Wahrheitswerte der Operanden als <i>UND</i> -Verbindung	x and y
or	binär	Disjunktion	verknüpft die Wahrheitswerte der Operanden als <i>ODER</i> -Verbindung	x or y

Logische Operatoren

a	b	not a	a and b	a or b
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

Logische Operatoren

Wenn andere Werte als der Datentyp *bool* mit den Operatoren **and** und **or** verknüpft werden, so wird immer einer der beiden Operanden zurückgegeben, wobei die Bedingungen für die Rückgabe wie folgt sind:

- [and]
 - Wenn der erste Operand falsch ist, wird dieser zurückgegeben.
 - Wenn der erste Operand wahr ist, wird der zweite Operand zurückgegeben.
- [or]
 - Wenn der erste Operand wahr ist, wird dieser zurückgegeben.
 - Wenn der erste Operand falsch ist, wird der zweite Operand zurückgegeben.

Logische Operatoren

```
>>> x,y = True,False
>>> not x
False
>>> not y
True
>>> x and y
False
>>> x or y
True
>>> x,y = False,False
>>> not x and not y
True
>>> x and y
False
>>> x or y
False
>>> x,y = 'harry','potter'
>>> x and y
'potter'
>>> x or y
'harry'
>>> x = False
>>> x and y
False
>>> x or y
'potter'
```

Relationale Operatoren

- Relationale Operatoren liefern immer einen Wahrheitswert in Bezug auf die Relation zurück, die überprüft werden soll.
- Bei den relationalen Operatoren kann zwischen Vergleichs-, Element- und Identitätsoperatoren unterschieden werden.

- Vergleichsoperatoren dienen dem Vergleich von Operanden hinsichtlich ihrer Werte.
- Zugehörigkeitsoperatoren dienen der Ermittlung der Zugehörigkeit eines Operanden zu anderen Operanden.
- Identitätsoperatoren dienen der Ermittlung von Identitätsverhältnissen, wobei Identität von Gleichheit ähnlich abgegrenzt wird wie dies im Deutschen mit Hilfe der Wörter *dasselbe* vs. *das gleiche* getan wird.

Relationale Operatoren

Operator	Klass.	Name	Erläuterung	Beispiel
<	binär	kleiner	prüft, ob ein Operand kleiner als der andere ist	x > y
>	binär	größer	prüft, ob ein Operand größer als der andere ist	x > y
==	binär	gleich	prüft, ob ein Operand gleich dem anderen ist	x == y
!=	binär	ungleich	prüft, ob ein Operand ungleich dem anderen ist	x != y
<=	binär	kleiner-gleich	prüft, ob ein Operand kleiner oder gleich dem anderen ist	x <= y
>=	binär	größer-gleich	prüft, ob ein Operand größer oder gleich dem anderen ist	x >= y
in	binär	Element von	prüft, ob ein Operand Element eines anderen ist	x in y
not in	binär	nicht Element von	prüft, ob ein Operand nicht das Element eines anderen ist	x not in y
is	binär	Identisch	prüft, ob ein Operand denselben Speicherort wie ein anderer besitzt	x is y
is not	binär	nicht Identisch	prüft, ob ein Operand nicht denselben Speicherort wie ein anderer besitzt	x is not y

Relationale Operatoren: Vergleichsoperatoren

```
>>> x,y = 5,10
>>> x > y
False
>>> x < y
True
>>> x == y
False
>>> x != y
True
>>> x <= y
True
>>> x >= y
False
```

Relationale Operatoren: Zugehörigkeitsoperatoren

```
>>> x,y = 'doculect','cul'  
>>> x in y  
False  
>>> y in x  
True  
>>> x not in y  
True
```

Relationale Operatoren: Identitätsoperatoren

```
>>> x = 1,2  
>>> y = x  
>>> x is y  
True  
>>> y = 1,2  
>>> x is y  
False
```

Arithmetische Operatoren

Die arithmetischen Operatoren in JavaScript sind im Großen und Ganzen identisch mit denen in Python. Lediglich der Potenz-Operator `**` existiert nicht.

Arithmetische Operatoren

```
> var x = 5; var y = 2;  
> +x;  
5  
> -x;  
-5  
> x + y;  
7  
> x - y;  
3  
> x * y;  
10  
> x / y;  
2.5  
> x // y;  
2  
> x % y;  
1  
> Math.pow(x,y); // x ** y in Python  
25
```

Logische Operatoren

Funktionell sind die logischen Operatoren in JavaScript identisch mit denen in Python, jedoch werden an Stelle der Schlüsselwörter

and, **or** und **not** in JavaScript die Zeichen

\&\&,

|| und ! verwendet. Ferner werden die beiden Werte des Datentyps **bool** klein geschrieben (**true** und **false** anstelle von **True** und **False**).

Logische Operatoren

```
> var x = true; var y = false
> !x; // not x in Python
false
> !y // not y in Python
true
> x && y // x and y in Python
false
> x || y // x or y in Python
true
> var x = false; var y = false;
> ! x && ! y // not x and not y in Python
true
> x && y
false
> x || y
false
> var x = 'harry'; var y = 'potter';
> x && y
'potter'
> x || y
'harry'
> var x = False
> var x && y
false
>>> x || y
'potter'
```

Relationale Operatoren

Auch die relationalen Operatoren in JavaScript sind denen in Python sehr ähnlich. Es fehlen jedoch die Operatoren **in** und **not in**. Anstelle der Schlüsselwörter **is** und **is not** werden die Symbole **==** und **!=** verwendet.

Relationale Operatoren: Vergleichsoperatoren

```
> var x = 5; var y = 10;  
> x > y  
false  
> x < y  
true  
> x == y  
false  
> x != y  
true  
> x <= y  
true  
> x >= y  
false
```

Relationale Operatoren: Vergleichsoperatoren

```
> var x = 'doculect'; var y = 'cul';  
> y.indexOf(x) != -1 // x in y in Python  
false  
> x.indexOf(y) != -1 // y in x in Python  
true  
> y.indexOf(x) == -1 // x not in y in Python  
true
```

Relationale Operatoren: Vergleichsoperatoren

```
> var x = [1, 2] // x = [1,2] in Python  
> var y = x  
> x === y // x is y in Python  
true  
> y = [1, 2]  
> x === y // x is y in Python  
false
```

5.2 Kontrollstrukturen

5.2.1 Allgemeines

Gängige Begriffserklärung aus Wikipedia

Kontrollstrukturen (Steuerkonstrukte) werden in imperativen Programmiersprachen verwendet, um den Ablauf eines Computerprogramms zu steuern. Eine Kontrollstruktur gehört entweder zur Gruppe der Verzweigungen oder der Schleifen. Meist wird ihre Ausführung über logische Ausdrücke der booleschen Algebra beeinflusst.

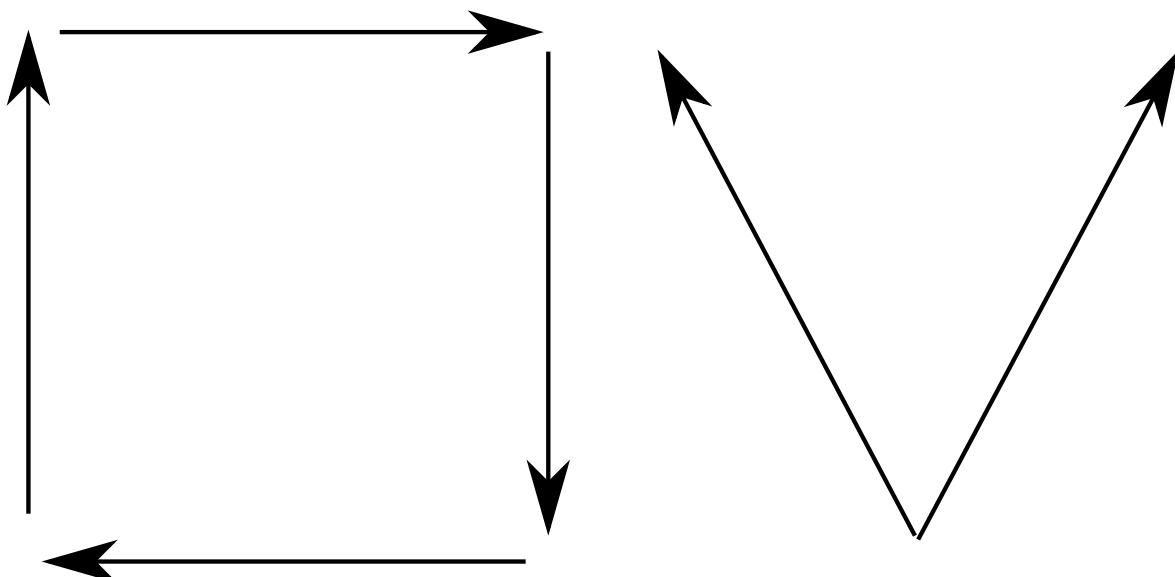
Begriffserklärung in Weigend (2008:127)

Kontrollstrukturen legen fest, in welcher Reihenfolge und unter welchen Bedingungen die Anweisungen eines Programms abgearbeitet werden.

Typen von Kontrollstrukturen

- **Verzweigungen** kontrollieren den Fluss eines Programms, indem sie dieses, abhängig von bestimmten Gegebenheiten, in unterschiedliche Bahnen lenken
- **Schleifen** kontrollieren den Fluss eines Programms, indem sie Operationen wiederholt auf Objekte anwenden

Typen von Kontrollstrukturen



5.2.2 ...in Python

Allgemeines

- Python kennt insgesamt vier verschiedene Kontrollstrukturen: zwei Verzweigungsstrukturen (**if**, **try**) und zwei Schleifenstrukturen (**for**, **while**).

- Das besondere an den Kontrollstrukturen in Python ist deren enge Anbindung an logische Ausdrücke, welche einen sehr kompakten, sehr leicht verständlichen Kode erlauben.

Verzweigungen: if, elif, und else

- Bei der **if**-Verzweigung wird ein Programmabschnitt unter einer bestimmten Bedingung ausgeführt.
- Dabei wird eine Bedingung auf ihren Wahrheitswert getestet. Trifft sie zu, wird das Programm entsprechend weitergeführt. Trifft sie nicht zu, unterbleibt der Abschnitt.
- Es können auch mehrere Bedingungen auf ihren Wahrheitswert überprüft und entsprechend mehrere verschiedene Programmabschnitte aktiviert werden.
- Es gibt in Python auch die Möglichkeit, *nichts* zu tun, wenn eine Bedingung zutrifft. Dies muss durch das Schlüsselwort **pass** festgelegt werden.

Verzweigungen: if, elif, und else

```
>>> x, y, yes, no = 10, 0, "yes", "no"
>>> if x: print(yes)
yes
>>> if y: print(yes)
>>> if not x: print(yes)
    elif not y: print no
no
>>> if 1 > 10: print("Eins ist grösser als zehn.")
    elif 1 < 10: print("Eins ist kleiner als zehn.")
Eins ist kleiner als zehn.
>>> if x or y: print("Einer von beiden Werten ist wahr.")
    else: print("Keiner von beiden Werten ist wahr.")
Einer von beiden Werten ist wahr.
>>> if x == 10: pass
    else: print("Danke, Herr Jauch!")
```

Verzweigungen: try und except

- Eine sehr wichtige und nützliche Verzweigungsstruktur bietet Python mit **try** und **except**.
- Hierbei wird nicht eine Bedingung auf ihren Wahrheitswert überprüft, sondern getestet, ob ein Statement eine Fehlermeldung hervorruft.
- Auf diese Weise kann man gezielt auf mögliche Fehler reagieren.
- Fehler können dabei gezielt entsprechend ihrer Klasse eingeordnet werden.

- Auf diese Weise können gezielt bestimmte Fehler angesprochen werden, die vom Programm ignoriert werden sollen.

Verzweigungen: try und except

```
>>> try: x = int(input())
       except: print("Der Wert, den Sie eingegeben haben, ist kein Integer!")
2
>>> try: x = int(input())
       except ValueError: print("Der Wert, den Sie eingegeben haben, ist kein Integer")
2.0
Der Wert, den Sie eingegeben haben, ist kein Integer!
>>> x = input()
10
>>> x, y = 10, 0
>>> try: x / y
       except ZeroDivisionError: print("Durch Null kann man nicht teilen!")
Durch Null kann man nicht teilen.
>>> try: x = int(input())
       except ValueError: print("Falscher Wert für Integer!")
10.0
Falscher Wert für Integer!
```

Schleifen: while

- Mit Hilfe von Schleifenkonstruktionen werden Blöcke von Anweisungen in Programmen wiederholt ausgeführt, bis eine Abbruchsbedingung eintritt, bzw. so lange eine Bedingung zutrifft.
- Wie für Verzweigungsstrukturen werden auch die Bedingungen in Schleifen auf Grundlage der Auswertung von Wahrheitswerten errechnet.
- Der Abbruch einer Schleife kann in Python bewusst durch das Schlüsselwort **break** gesteuert werden.

Schleifen: while

```
>>> a = 10
>>> while a > 0:
       print(a, " ist grösser als 0.")
       a -= 1 # wir lassen der Wert gegen 0 laufen

10 ist grösser als 0
9 ist grösser als 0
8 ist grösser als 0
7 ist grösser als 0
```

```

6 ist groesser als 0
5 ist groesser als 0
4 ist groesser als 0
3 ist groesser als 0
2 ist groesser als 0
1 ist groesser als 0
>>> while True:
    b = input("Sag was: ")
    print b
    if not b:
        print("Aaaaaabbbbrrrrruuuucccccch!")
        break
Sag was: bla
bla
Sag was: bloed
bloed
Aaaaaabbbbrrrrruuuucccccch!

```

Schleifen: for

- Während bei der **while**-Schleife die Abbruchsbedingung immer explizit genannt werden muss, gilt dies nicht für die **for**-Schleife.
- Hier wird explizit und im Voraus festgelegt, wie oft, bzw. in Bezug auf welche Objekte eine bestimmte Operation ausgeführt werden soll, weshalb sich die **for**-Schleife anbietet, wenn man die Anzahl der Operationen kennt, die man durchführen will.
- Der Bereich, in dem in Python eine Operation mit Hilfe der **for**-Schleife ausgeführt wird, wird zumeist mit Hilfe der Funktion **range()** angegeben, welche automatisch eine Liste von Integern erzeugt, über die dann iteriert wird.

Schleifen: for

```

>>> a = range(5)
>>> a
range(0,5)
>>> list(a)
[0, 1, 2, 3, 4]
>>> for i in range(5): print(i)
0
1
2
3
4
>>> einkaufsliste = ['mehl','butter','zitronen','bier']
>>> for element in einkaufsliste: print element

```

```

mehl
butter
zitronen
bier
>>> namen = ['peter','frank','karl','jan']
>>> for name in namen: print(name)
peter
frank
karl
jan
>>> woerter = ['haus','kaffee','getraenk','wasser','flasche']
>>> for wort in woerter: print(wort)
haus
kaffee
getraenk
wasser
flasche

```

5.2.3 ... in JavaScript

Verzweigungen: if, else if und else

Die if-else-Verzweigung in JavaScript unterscheidet sich in ihrer Syntax von Python, nicht jedoch in der grundlegenden Konzeption, auf der sie aufbaut. Auch in JavaScript wird der Wahrheitswert einer Datenstruktur getestet und entsprechend reagiert. Anstelle von "elif" wird dabei in JavaScript "else if" geschrieben. Ferner werden hässliche Klammern verwendet anstelle der schönen Einrückungen, die in Python verwendet werden.

Verzweigungen: if, else if und else

```

> var x = 10; var y = 0; var yes = 'yes'; var no = 'no';
> if (x) {console.log(yes);} // if x: print(yes) in Python
yes
> if (y) {console.log(yes);} // if y: print(yes) in Python
>>> if (!x) {console.log(yes);}
    else if(!y) { console.log(no);}
no
>>> if (1 > 10) {console.log("Eins ist groesser als zehn.");}
    else if (1 < 10) {console.log("Eins ist kleiner als zehn.");}
Eins ist kleiner als zehn.
>>> if (x || y) {console.log("Einer von beiden Werten ist wahr.");} // if x or y
    else {console.log("Keiner von beiden Werten ist wahr.");}
Einer von beiden Werten ist wahr.
>>> if (x == 10) {}
    else {console.log("Danke, Herr Jauch!");}

```

Verzweigungen: try und catch

Anstelle von **try** und **except** bietet JavaScript die Verzweigung **try** und **catch** an. Die Syntax ist auch hier wieder anders als in Python, die grundlegende Struktur ist jedoch sehr ähnlich. Allerdings sollte man diese Verzweigung nach Möglichkeit nur spärlich anwenden, da eine Vielzahl von Fehlern in JavaScript nicht als solche definiert werden. So führt eine Division durch 0 beispielsweise nicht zu einem ZeroDivisionError, sondern zum Wert *Infinity*. Ferner führt der Befehl `parseInt('m')` nicht zu einem ValueError, sondern zum Wert *NaN*. Aus diesem Grund werden

try und **catch** hier nicht weiter behandelt.

Schleifen: while

Abgesehen von der Verwendung der hässlichen Klammerstrukturen funktioniert die **while**-Schleife in JavaScript im Prinzip genauso wie die **while**-Schleife in Python. Auch das Schlüsselwort **break** kann verwendet werden.

Schleifen: while

```
> var a = 10
> while (a > 0) { console.log(a+" ist groesser als 0."); a -= 1;}
10 ist groesser als 0
9 ist groesser als 0
8 ist groesser als 0
7 ist groesser als 0
6 ist groesser als 0
5 ist groesser als 0
4 ist groesser als 0
3 ist groesser als 0
2 ist groesser als 0
1 ist groesser als 0
> var a = 10;
> while(true){a -= 1; console.log(a+' ist groesser als 0'); if(a == 0){break;}}
9 ist groesser als 0
8 ist groesser als 0
7 ist groesser als 0
6 ist groesser als 0
5 ist groesser als 0
4 ist groesser als 0
3 ist groesser als 0
2 ist groesser als 0
1 ist groesser als 0
0 ist groesser als 0
```

Schleifen: for

In Bezug auf die **for**-Schleife unterscheiden sich Python und JavaScript ein

wenig mehr als in Bezug auf die anderen Kontrollstrukturen. Während die **for**-Schleife in Python generell auf der Datenstruktur von iterierbaren Elementen (vorrangig Listen) aufbaut, wird in JavaScript eine explizite Zählerstruktur benötigt. Zwar gibt es auch ein **for value in iterable**-Konstrukt in JavaScript, jedoch gibt dies nur die Schlüsselwerte von Objekten (**dictionaries** in Python) aus und sollte nie zusammen mit Arrays verwendet werden.

Schleifen: for

```
> var liste = ["a", "b", "c", "d"];
> for (var i=0; i < liste.length; i++) {console.log(i, liste[i]);}
0 'a'
1 'b'
2 'c'
3 'd'
> for (var i=0,elm; elm=liste[i]; i++) {console.log(i, elm);}
0 'a'
1 'b'
2 'c'
3 'd'
> var myobj = {"a":1, "b":2, "c":3, "d":4};
> for (key in myobj) {console.log(key, myobj[key]);}
a 1
b 2
c 3
d 4
```

6 Funktion

6.1 Funktionen im Allgemeinen

6.1.1 Begriffserklärung

Funktionen in der Mathematik (laut Wikipedia)

In der Mathematik ist eine Funktion oder Abbildung eine Beziehung zwischen zwei Mengen, die jedem Element der einen Menge (Funktionsargument, unabhängige Variable, x-Wert) genau ein Element der anderen Menge (Funktionswert, abhängige Variable, y-Wert) zuordnet. Das Konzept der Funktion oder Abbildung nimmt in der modernen Mathematik eine zentrale Stellung ein; es enthält als Spezialfälle unter anderem parametrische Kurven, Skalar- und Vektorfelder, Transformationen, Operationen, Operatoren und vieles mehr.

Funktionen in der Programmierung (laut Wikipedia)

Eine Funktion (engl.: function, subroutine) ist in der Informatik die Bezeichnung eines Programmierkonzeptes, das große Ähnlichkeit zum Konzept der Prozedur hat. Hauptmerkmal einer Funktion ist es, dass sie ein Resultat zurückliefert und deshalb im Inneren von Ausdrücken verwendet werden kann. Durch diese Eigenschaft grenzt sie sich von einer Prozedur ab, die nach ihrem Aufruf kein Ergebnis/Resultat zurück liefert. Die genaue Bezeichnung und Details ihrer Ausprägung ist in verschiedenen Programmiersprachen durchaus unterschiedlich.

6.1.2 Typen von Funktionen

Da entscheidend für Funktionen der Eingabewert und der Rückgabewert sind, können wir ganz grob die folgenden speziellen Typen von Funktionen identifizieren:

- Funktionen ohne Rückgabewert (Prozedur)
- Funktionen ohne Eingabewert
- Funktionen mit einer festen Anzahl von Eingabewerten
- Funktionen mit einer beliebigen Anzahl von Eingabewerten

i

6.2 Funktionen in Python

Vorweg

In Python sind Funktionen spezielle Datentypen. Sie unterscheiden sich von anderen Datentypen wie **strings** oder **integers** dadurch, dass sie zusätzlich aufgerufen werden können (*callable types*).

6.2.1 Grundlagen

Allgemeine Struktur

```
def functionName(  
    parameterA,  
    ...  
    keywordA='defaultA',  
    ...  
):  
    """  
    docString  
    """  
    functionBody  
    return functionValue
```

Erstes Beispiel

```

def stoneAgeCalculator(intA, intB, calc='+'):
    """
    This is the famous stoneAgeCalculator, a program written by the very first
    men on earth who brought the fire to us and were the first to dance naked
    on the first of May.
    """
    # check for inconsistencies in the input for keyword calc
    if calc not in ['+', '-', '*', '/']:
        raise ValueError('The operation you chose is not defined.')

    # start the calculation, catch errors from input with a simple
    # try-except-structure
    try:
        if calc == '+':
            return intA + intB
        elif calc == '-':
            return intA - intB
        elif calc == '*':
            return intA * intB
        else:
            return intA / intB
    except:
        raise ValueError('No way to operate on your input.')

```

Aufrufen und Ausführen

Eine Funktion wird aufgerufen, indem der Name der Funktion angegeben wird, gefolgt von einer Liste aktueller Parameter in Klammern. (Weigend 2008:144)

```

>>> a = range(5)
>>> a = int('1')
>>> range(5)
range(0,5)
>>> int('1')
1
>>> float(1)
1.0
>>> print('Hallo Welt!')
Hallo Welt!
>>> list('hallo')
['h', 'a', 'l', 'l', 'o']

```

Die Funktionsdokumentation

Für jede Funktion, die in Python definiert wird, steht eine automatische Dokumentation bereit, welche mit Hilfe der Parameter und der Angaben im Docstring generiert wird. Die Dokumentation einer Funktion wird mit Hilfe von

`help(function)` aufgerufen. Mit Hilfe der Funktionsdokumentation lässt sich leicht ermitteln, welche Eingabewerte eine Funktion benötigt, wie sie verwendet wird, und welche Werte sie zurückliefert. Docstrings lassen sich am besten interaktiv einsehen.

6.2.2 Parameter und Schlüsselwörter

Parameter

Mit Hilfe von Funktionen werden Eingabewerte (Parameter) in Ausgabewerte umgewandelt. Bezuglich der Datentypen von Eingabewerten gibt es in Python keine Grenzen. Alle Datentypen (also auch Funktionen selbst) können daher als Eingabewerte für Funktionen verwendet werden. Gleichzeitig ist es möglich, Funktionen ohne Eingabewerte oder ohne Ausgabewerte zu definieren. Will man eine beliebige Anzahl an Parametern als Eingabe einer Funktion verwenden, so erreicht man dies, indem man der Parameterbezeichnung einen Stern (*) voranstellt. Die Parameter von Eingabeseite werden innerhalb der Funktion dann automatisch in eine Liste umgewandelt, auf die mit den normalen Listenoperationen zugegriffen werden kann.

Parameter

```
>>> def leer():
...     print "leer"
>>> def empty():
...     pass
>>> def gruss1(wort):
...     print wort
>>> def gruss2(wort):
...     return wort
>>> def eins():
...     """Gibt die Zahl 1 aus."""
...     return 1
>>> leer()
leer
>>> empty
...
>>> empty()
>>> type(empty)

>>> a = eins()
>>> a
1
>>> print bla.func_doc
Gibt die Zahl 1 aus.
>>> def printWords(*words):
...     for word in words:
...         print(word)
```

```

...
>>> printWords('mama', 'papa', 'oma', opa')
mama
papa
oma
opa

```

Schlüsselwörter

Als Schlüsselwörter bezeichnet man Funktionsparameter, die mit einem Standardwert belegt sind. Werden diese beim Funktionsaufruf nicht angegeben, gibt es keine Fehlermeldung, sondern anstelle der fehlenden Werte wird einfach auf den Standardwert zurückgegriffen. Gleichzeitig braucht man beim Aufrufen nicht auf die Reihenfolge der Parameter zu achten, da diese ja durch die Anbindung der Schlüsselwörter vordefiniert ist. Weist eine Funktion hingegen Schlüsselwörter und Parameter auf, so müssen die Parameter den Schlüsselwörtern immer vorangehen.

Schlüsselwörter

```

>>> def wind(country, season='summer')
...     """Return the typical wind conditions for a given country."""
...     if country == "Germany" and season == 'summer':
...         print("There's strong wind in",country)
...     else:
...         print("This part hasn't been programmed yet.")
...
>>> wind('Germany'):
There's strong wind in Germany.
>>> wind('Germany',season='winter')
This part hasn't been programmed yet.

```

6.2.3 Spezialfälle

Prozeduren

Prozeduren sind Funktionen, die den Wert **None** zurückgeben. Fehlt in einer Funktionsdefinition die **return**-Anweisung, wird automatisch der Wert **None** zurückgegeben. (Weigend 2008: 155)

```

>>> def quak(frosch=''):
...     print("quak")
...
>>> quak()
quak
>>> a = quak()

```

```
quak
>>> type(a)
```

Rekursive Funktionen

Rekursive Funktionen sind solche, die sich selbst aufrufen. (Weigend 2008: 151)

```
>>> def factorial(number):
...     """
...     Aus: http://www.dreamincode.net/code/snippet2800.htm
...     """
...     if number == 0:
...         return 1
...     else:
...         value = factorial(n-1)
...         result = n * value
...         return result
...
>>> factorial(4)
24
```

Globale und lokale Variablen

Es muss bei Funktionen zwischen lokalen und globalen Variablen unterschieden werden. Lokale Variablen haben nur innerhalb einer Funktion ihren Gültigkeitsbereich. Globale Variablen hingegen gelten auch außerhalb der Funktion. Will man mit Hilfe einer Funktion eine Variable, die global, also außerhalb der Funktion deklariert wurde, verändern, so muss man ihr das *Keyword* **global** voranstellen. Dies sollte man jedoch nach Möglichkeit vermeiden, da dies schnell zu Fehlern im Programmablauf führen kann. Lokale und globale Variablen sollten nach Möglichkeit getrennt werden.

Globale und lokale Variablen

```
>>> s = 'globaler string'
>>> def f1():
...     print(s)
...
>>> def f2():
...     s = 'lokaler string'
...     print(s)
...
>>> f1()
globaler string
>>> f2()
```

```

lokaler string
>>> def f3():
...     global s
...     s = 'lokaler string'
...     print(s)
>>> s = 'globaler string'
>>> f3()
lokaler string
>>> print(s)
lokaler string

```

6.3 Funktionen in JavaScript

6.3.1 Grundlagen

Allgemeines vorweg

Wie in Python, so sind auch in JavaScript Funktionen spezielle Datentypen.

Allgemeine Struktur

```

function functionName (
    parameterA,
    parameterB,
    parameterC,
    ...
) {
    functionBody;
    return functionValue;
}

```

Erstes Beispiel: Der Steinzeittaschenrechner

```

function stoneAgeCalculator(intA, intB, calc) {
    /* This is a very famous StoneAge Calculator. */

    /* check for active values for type */
    if (typeof calc == 'undefined') {
        calc = "+";
    }
    /* check for correct values for calc */
    else if(["+", "-", "*", "/"].indexOf(calc) == -1) {
        return false;
    }

```

```

/* return the stuff */
if (calc == '+') {return intA + intB;}
else if (calc == '-') {return intA - intB;}
else if (calc == '*') {return intA * intB;}
else if (calc == '/') {return intA / intB;}
return false;
}

```

Erstes Beispiel: Der Steinzeittaschenrechner

6.3.2 Parameter und Schlüsselwörter

Parameter und Schlüsselwörter

Die Handhabung von Parametern und Schlüsselwörtern ist nicht so leicht in JavaScript wie in Python. Schlüsselwörter gibt eigentlich gar nicht. Das heißt leider auch, dass man keine Standardwerte für bestimmte Parameter annehmen kann, was das Programmieren ein wenig umständlich macht. Um ähnliche Funktionalitäten wie in Python zu erlangen, behilft man sich meist mit Ersatzkonstruktionen.

Parameter und Schlüsselwörter

```

function wind(country, season) {
    /* Return the typical wind conditions for a country */

    /* carry out type check of season, to set the basic value */
    if (typeof season == "undefined") {
        season = "summer";
    }

    if (country == "Germany" and season == "summer") {
        return "There's strong wind in "+country+"." ;
    }
    else {
        return "This part hasn't been programmed yet.";
    }
}

```

Parameter und Schlüsselwörter

6.3.3 Spezialfälle

Prozeduren und Rekursive Funktionen

Prozeduren und rekursive Funktionen können in JavaScript genauso verwendet werden wie in Python. Tendenziell werden sehr viel mehr Prozeduren in JavaScript verwendet, da das Auslösen von Aktionen mit Hilfe der HTML-Buttons und der “onclick”-Anweisung am leichtesten geregelt werden kann.

Globale und lokale Variablen

JavaScript unterscheidet nicht direkt zwischen globalen und lokalen Variablen: Alles, was auf einer jeweils höheren Ebene definiert wurde, kann auch auf einer niedrigeren Ebene verwendet werden und umgekehrt. Das Schlüsselwort **var** kann jedoch verwendet werden, um zu gewährleisten, dass Variablen nur innerhalb ihrer jeweiligen Ebene Geltung haben. Sicherheitshalber sollte man daher vor jede Variablendeklaration das Schlüsselwort **var** setzen.

Globale und lokale Variablen

```
> s = 'globaler string'
> function f1(){console.log(s)}
> function f2(){s = 'lokaler string'; console.log(s);}
> f1()
'globaler string'
> f2()
'lokaler string'
> function f3() { var s; s = 'lokaler string'; console.log(s);}
> s = 'globaler string';
> f3()
lokaler string
> s
globaler string
```

7 Sequenzvergleiche in der historischen Linguistik

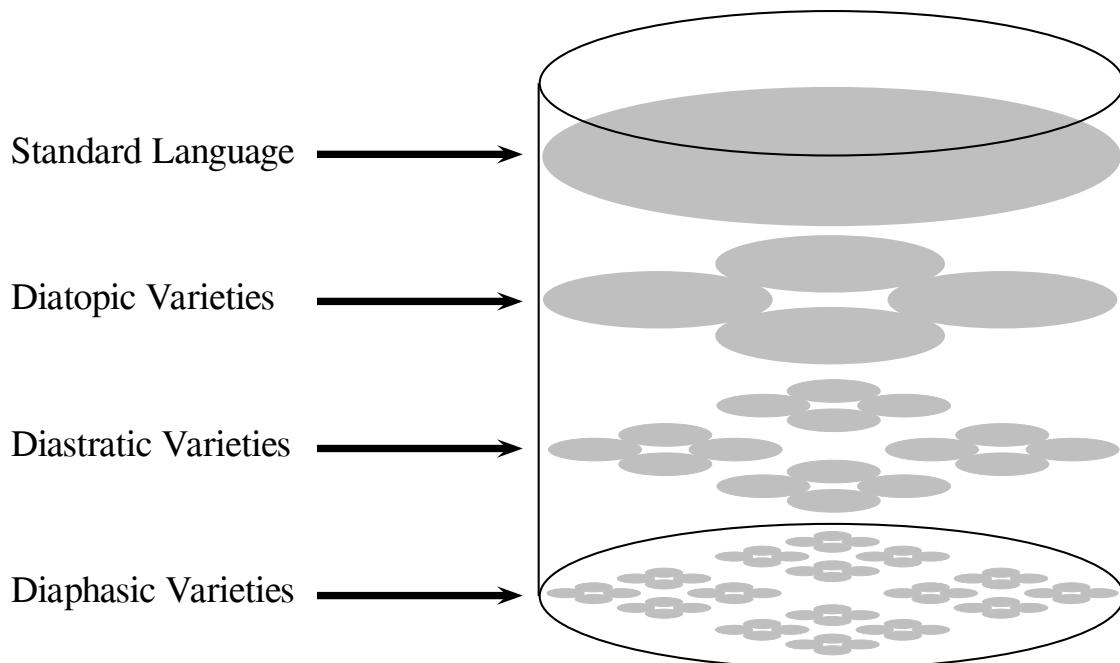
7.1 Sprachwandel

7.1.1 Sprachen

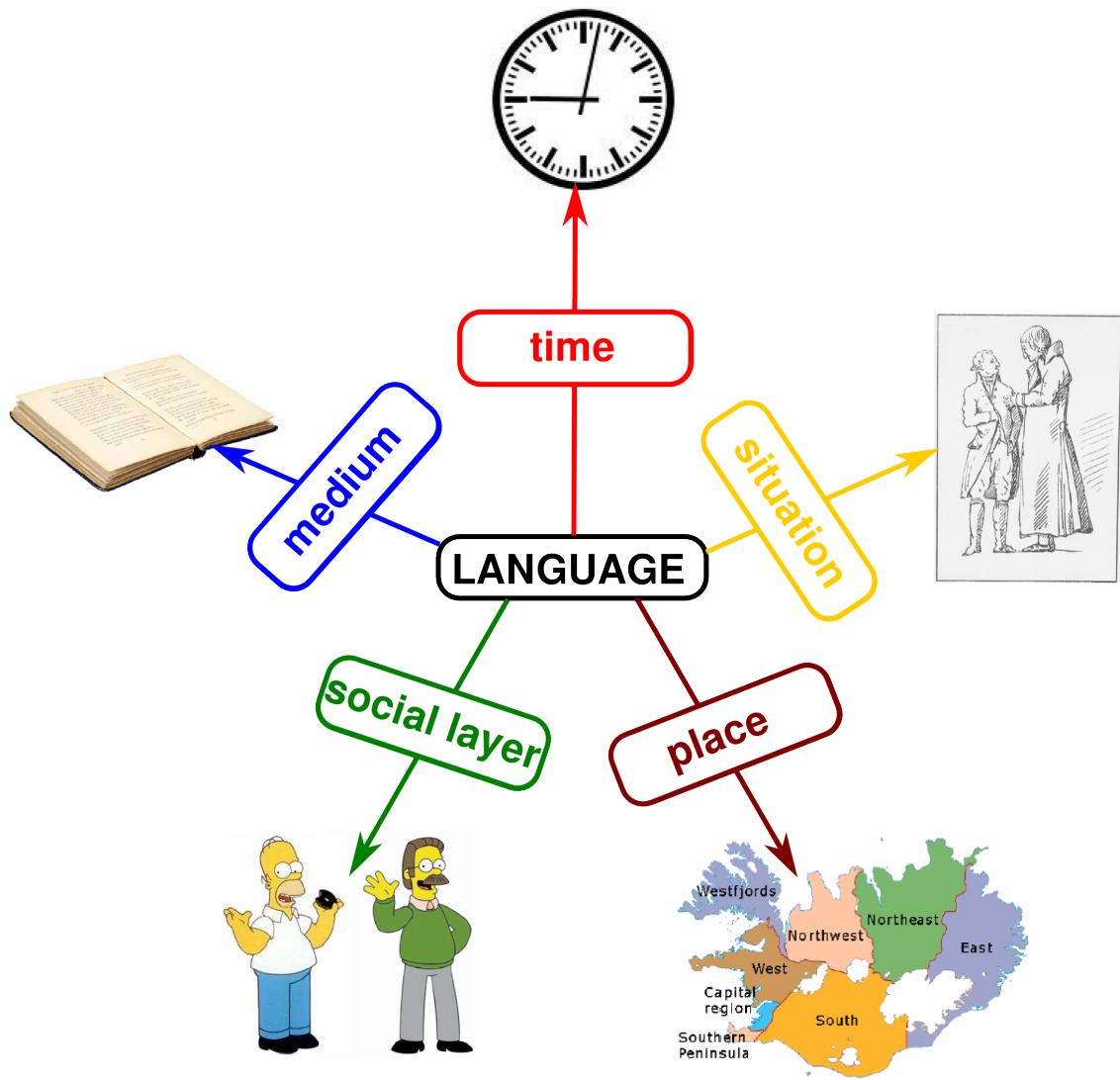
Der Nordwind und die Sonne und das Problem der Sprachgrenzen

Bēijīng Chinese	1	iou ²¹	i ⁵⁵	xuei ³⁵	pei ²¹ fəŋ ⁵⁵	kən ⁵⁵	tʰai ⁵¹ iaŋ ¹¹	tʂəŋ ⁵⁵	tsai ⁵³	naə ⁵¹	tʂəŋ ⁵⁵ luən ⁵¹
Hakka Chinese	1	iu ³³	it ⁵⁵	pai ³³ a ¹¹	pet ³³ fuiŋ ³³	tʰuŋ ¹¹	jiit ¹¹ tʰeu ¹¹	hɔk ³³	e ⁵³	au ⁵⁵	
Shànghǎi Chinese	1	fi ²²		tʰɑ ⁵⁵ tsɿ ²¹	po ³³ fɔŋ ⁴⁴	ta? ⁵	tʰa ³³ fia ⁴⁴	tsəŋ ³³ hɔ ⁴⁴		lə ²¹ lə ²³ tsa ⁵³	
		was one	time	northwind and		sun		moment	there	discuss	
Bēijīng Chinese	2	sei ³⁵		də ⁵⁵		pən ³⁵ lin ²¹	ta ⁵¹				
Hakka Chinese	2	man ³³	ɲin ¹¹			kʷɔ ⁵⁵	vɔi ⁵³				
Shànghǎi Chinese	2	sa ³³	ɲin ⁵⁵	fiə? ²¹		pəŋ ³³ zɿ ⁴⁴	du ¹³				
		who	person of	exceed power		big					
Norwegian	1	nu:ravɪn'ɳ	ɔ	su:lɳ						krɑŋlət	ɔm
Swedish	1	nu:ðanvindən	ɔ	su:lən		ty̯istadə	ən gɔj			ɔm	
Danish	1	nøðʌnven?ɳ	ʌ	so:l?n	kʰam	en̩gæŋ	i sðøið?			ʌm?	
		northwind	and	sun	come argue	one time in fight	fight			about	
Norwegian	2	vem	a	dem	sŋ	və:	dŋ			stærkəstə	
Swedish	2	vem	av	dɔm	sɔm	və	dŋ			staukast	
Danish	2	vem?	a	bm	d	və	dŋ			sðæʌg̩esdə	
		who	of	them	who was	the	the			strongest	

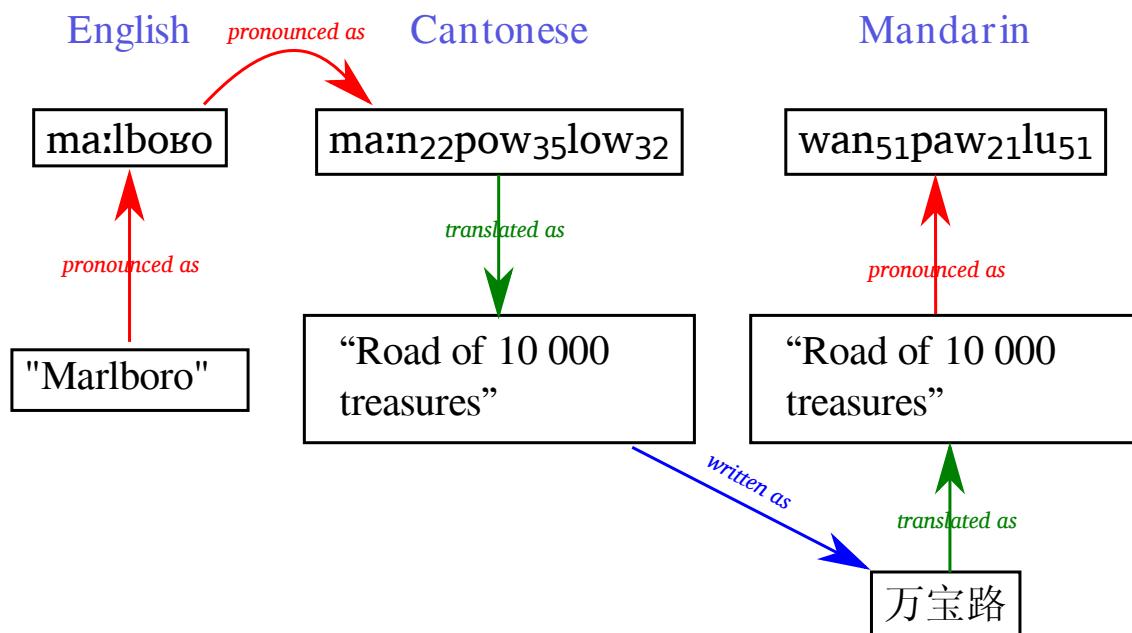
Diasysteme



Dimensionen der Sprachvariation



Komplexität des Sprachwandels



7.1.2 Sprachwandel im Allgemeinen

Die komischen Reime in den chinesischen Oden

燕	燕	于	飛,	下	上	其	音。	The swallows go flying, falling and rising are their voices;
yān	yān	yú	fēi	xià	shàng	qí	yīn	
之	子	于	歸,	遠	送	于	南。	This young lady goes to her new home, far I accompany her to the south
zhī	zǐ	yú	guī,	yuǎn	sòng	yú	nán	
瞻	望	弗	及,	實	勞	我	心。	I gaze after her, can no longer see her, truly it grieves my heart.
zhān	wàng	fú	jí,	shí	láo	wǒ	xīn	

Die komischen Reime in den chinesischen Oden

The writings of scholars must be made of adequate sounds. Even in the rural areas everybody orders the sounds harmonically. Can it be that the ancients solely did not have rhymes? One can say that in the same way in which ancient times differ from modern times, and places in the North differ from places in the South, characters change and sounds shift. This is a natural tendency. Therefore, it is inevitable that reading the ancient writings with modern pronunciation will sound improper and wrong. (Máoshī Gǔyīnkǎo: 古今考, meine Übersetzung)

Wandel als Katastrophe

Schon früh in der Geschichte der Linguistik war den Forschern in Europa bewusst, dass Sprachen sich wandeln können. Vorherrschend war dabei jedoch die

Ansicht, dass alle Formen des Wandels „katastrophisch“ abliefen, dass Wandel also im Rahmen eines unberechenbaren, chaotischen „Verfalls“ vor sich ginge. Erst spät (zu Beginn des 19. Jahrhunderts) wurde erstmals klar, dass sich bestimmte Phänomene des Sprachwandels, insbesondere der Lautwandel, durch eine beachtliche Regelmäßigkeit auszeichnen.

Wandel als Prozess

Meaning	Italian		Latin	
	Orth.	IPA	Orth.	IPA
‘key’	<i>chiave</i>	kjave	<i>clāvis</i>	kla:wis
‘feather’	<i>piuma</i>	pjuma	<i>plūma</i>	plu:ma
‘flower’	<i>fiore</i>	fjore	<i>flōs</i>	f o:s
‘tear’	<i>lacrima</i>	lakrima	<i>lacrima</i>	lakrima
‘tongue’	<i>lingua</i>	lingwa	<i>lingua</i>	lingwa
‘moon’	<i>luna</i>	luna	<i>lūna</i>	lu:na

Wandel und Regelmäßigkeit

Während den Menschen im Verlaufe der Geschichte bereits relativ lange bewusst war, dass Sprachen sich ändern können, war es eine radikal neue Erkenntnis, die sich zu Beginn des 19. Jahrhunderts herauskristallisierte, dass Sprachen sich in Prozessen ändern, von denen bestimmte sogar regelmäßig verlaufen können. Mit der Entdeckung der Regelmäßigkeit einher festigte sich ebenfalls die Erkenntnis, dass Sprachen miteinander verwandt sein können, wobei Verwandtschaft von Sprachen dadurch definiert ist, dass miteinander verwandte Sprachen aus einer gemeinsamen Vorgängersprache entstanden sind, wie bspw. das Englische und das Deutsche, die beide aus dem Protogermanischen hervorgegangen sind.

7.1.3 Lautwandel

Wandel als Gesetz (Osthoff und Brugmann 1878: XIII)

Aller lautwandel, soweit er mechanisch vor sich geht, vollzieht sich nach ausnahmslosen gesetzen, d.h. die richtung der lautbewegung ist bei allen angehörigen einer sprachgenossenschaft, ausser dem Fall, dass dialektspaltung eintritt, stets dieselbe, und alle wörter, in denen der der lautbewegung unterworfenen laut unter gleichen verhältnissen erscheint, werden ohne ausnahme von der änderung ergriffen.

Chaque mot a son histoire

Nicht alle Linguisten waren der Meinung der Junggrammatiker. Besonders Dialektologen folgten dem berühmten Slogan *chaque mot a son histoire* ("jedes Wort hat seine Geschichte"), der gewöhnlich Jules Gilliéron (1854–1926) zugeschrieben wird (Campbell 1999: 189). Die Bedenken der Dialektologen standen jedoch streng genommen nicht direkt im Widerspruch zur junggrammatischen Doktrin, schließlich besagte die junggrammatische Theorie ja nicht, dass sich zwangsläufig *alle* Wörter einer Sprache regelmäßig änderten, sondern lediglich, dass idiosynkratischer Wandel durch andere Mechanismen (Entlehnung oder Analogie) erklärt werden konnte (Kiparsky 1988:368).

7.2 Lautwandel

7.2.1 Mechanismen des Lautwandels

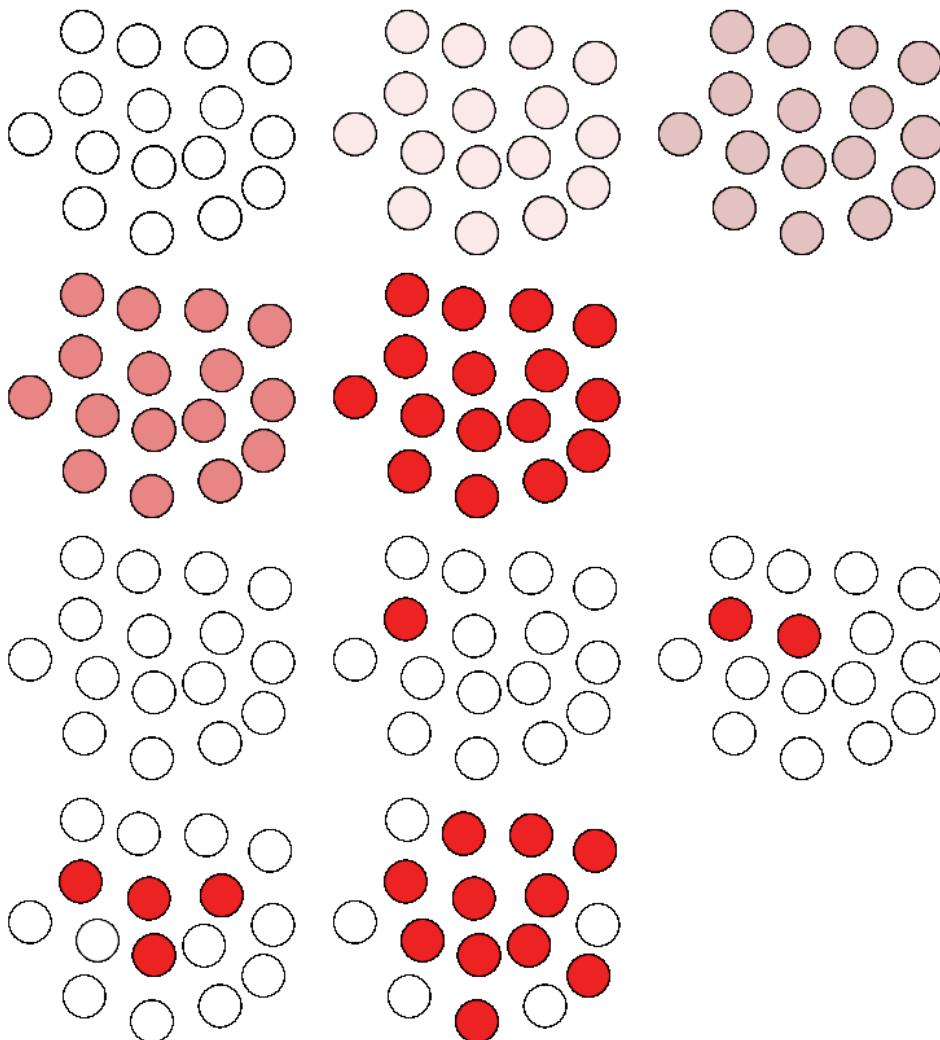
Junggrammatischer Lautwandel (nach Wang 2006: 109, meine Übersetzung)

Regarding the lexicon [they assumed] that a change always affects the whole lexicon, and can therefore be seen as an abrupt change. Regarding the sounds [they assumed] that the change proceeded step by step, and can therefore be seen as a gradual change.

Lexikalische Diffusion (Wang 1969: 9)

Phonological change may be implemented in a manner that is phonetically abrupt but lexically gradual. As the change diffuses across the lexicon, it may not reach all the morphemes to which it is applicable. If there is another change competing for part of the lexicon, residue may result.

Kompetitierende Theorien der Lautwandelmechanismen



Zwei Mechanismen des Lautwandels (Labov 1994: 471)

There is no basis for contending that lexical diffusion is somehow more fundamental than regular, phonetically motivated sound change. On the contrary, if we were to decide the issue by counting cases, there appear to be far more substantially documented cases of Neogrammarian sound change than of lexical diffusion.

7.2.2 Typen des Lautwandels

Terminologische Probleme

Die lange Forschungstradition in der historischen Linguistik hat zur Postulierung einer Vielzahl unterschiedlicher Typen des Lautwandels} geführt. Leider ist die Terminologie, welche verwendet wird, um auf diese Typen in der Literatur zu verweisen, etwas “unstetig” und reicht von sehr konkreten Termini, welche sehr konkrete Lautwandelinstanzen abdecken, bis hin zu sehr generellen Termini, die auf den Wandel abstrakter Lautklassen verweisen.

Terminologische Probleme

Was in der Literatur als *Lautwandeltyp* bezeichnet wird, kann dabei sowohl das Phänomen des *Rhotazismus* umfassen (Trask 2000: 288), welches, vereinfacht gesagt, einen Wandel von /s/ nach /r/ bezeichnet, als auch den Prozess der *Lenisierung*, welcher eine bestimmte Art von Wandel bezeichnet, “in which a segment becomes less consonant-like than previously” (idb. 190).

Definitionen von Trask (2000)

- **Assimilierung** “Any **syntagmatic change** in which some segment becomes more similar in nature to another segment in the same sequence, usually within a single phonological word or phrase” (30).
- **Dissimilierung** “Any **syntagmatic change** in which one segment changes so as to become less similar to another segment in the same form” (95).
- **Metathese** “Any **syntagmatic change** in which the order of segments (or sometimes of other phonological elements) in a word is altered” (211).
- **Tonogenese** “Any process which leads to the introduction of tones into a language which formerly lacked them” (346).

Definitionen von Trask (2000, cont.)

- **Sandhi** “Any of various phonological processes applying to sequences of segments either across morpheme boundaries (*internal sandhi*) or across word boundaries (*external sandhi*)” (296).
- **Haplologie** “A type of phonological change (of or phonological constraint) in which one of two adjacent syllables of identical or similar form is lost (or fails to appear in the first place)” (146).
- **Elision (Aphaerese, Synkope, Apokope)** “Any of various processes in which phonological segments are lost from a word or a phrase. Specific varieties of elision are often given special names like **aphaeresis, syncope, apocope, synaeresis, synizesis, synaloepha**. Not infrequently this name is given to specific processes in particular languages” (102).

Definitionen von Trask (2000, cont.)

- **Epenthese** “Any phonological change which inserts a segment into a word or form in a position in which no segment was formerly present” (107).
- **Prothése** “The addition of a segment to the beginning of a word. [...] The opposite is **aphaeresis**” (266).
- **Nasalierung** “Any phonological process in which a segment acquires a nasal character which it formerly lacked” (224).

7.2.3 Typen des Lautwandels

Ein vereinfachtes Modell

Type	Description	Representation
continuation	absence of change	$x > x$
substitution	replacement of a sound	$x > y$
insertion	gain of a sound	$\emptyset > y$
deletion	loss of a sound	$x > \emptyset$
metathesis	change in the order of sounds	$xy > yx$

7.3 Lautwandel

7.3.1 Typen des Lautwandels

Ein vereinfachtes Modell (Beispiele)

Input	Output	Typ
Urslavisch *žltb 'gelb'	Czech žlutý [ʒluti:] 'gelb' (DERKSEN: 565)	metathesis
Althochdeutsch angust [aŋust]	Hochdeutsch Angst [aŋst]	deletion
Althochdeutsch hant [hant]	Hochdeutsch Hand [hant]	continuation
Althochdeutsch ioman [jo-man]	Hochdeutsch jemand [je-mant]	insertion
Althochdeutsch snēo [sne:o]	Hocheutsch Schnee [ʃne:]	substitution

7.3.2 Die komparative Methode

Definitionen

→ comparative linguistics, **reconstruction**

Routledge Dictionary of Language and Linguistics
(Bussmann 1996)

In linguistics, the comparative method is a **technique for studying the development of languages** by performing a feature-by-feature comparison of two or more languages with common descent from a shared ancestor, as **opposed to the method of internal reconstruction**, which analyses the internal development of a single language over time.

Wikipedia s.v. "Comparative Method"

The Comparative Method is **the central tool in historical linguistics** for historical reconstruction **and also classifying languages**. A classification done with the Comparative Method is called a genetic classification. The result is that languages are arranged in language family trees. This means that languages are classified according to their genealogical relationships² and are interpreted as being in relation of child- or sisterhood to other languages. Such a way of classifying entities is called phylogenetic classification in biology; a classification by genealogical relationships.

Fleischhauer (2009)

The method of **comparing languages to determine whether and how they have developed from a common ancestor**. The items compared are lexical and grammatical units, and the aim is to discover correspondences relating sounds in two or more different languages, which are so numerous and so regular, across sets of units with similar meanings, that no other explanation is reasonable.

Oxford Dictionary of Linguistics (Matthews 1997)

The method of comparatistics today is generally known under the not very well-chosen term "comparative-historical method". It constitutes a **huge complex of abstract and concrete procedures for the investigation of the history of related languages** which genetically go back to some unfrom tradition of the past.

Klimov (1990), my translation

The comparative is both the earliest and **the most important of the methods of reconstruction**. Most of the major insights into the prehistory of languages have been gained by the applications of this method, and most reconstructions have been based on it.

Fox (1995)

Ein neuer Definitionsversuch

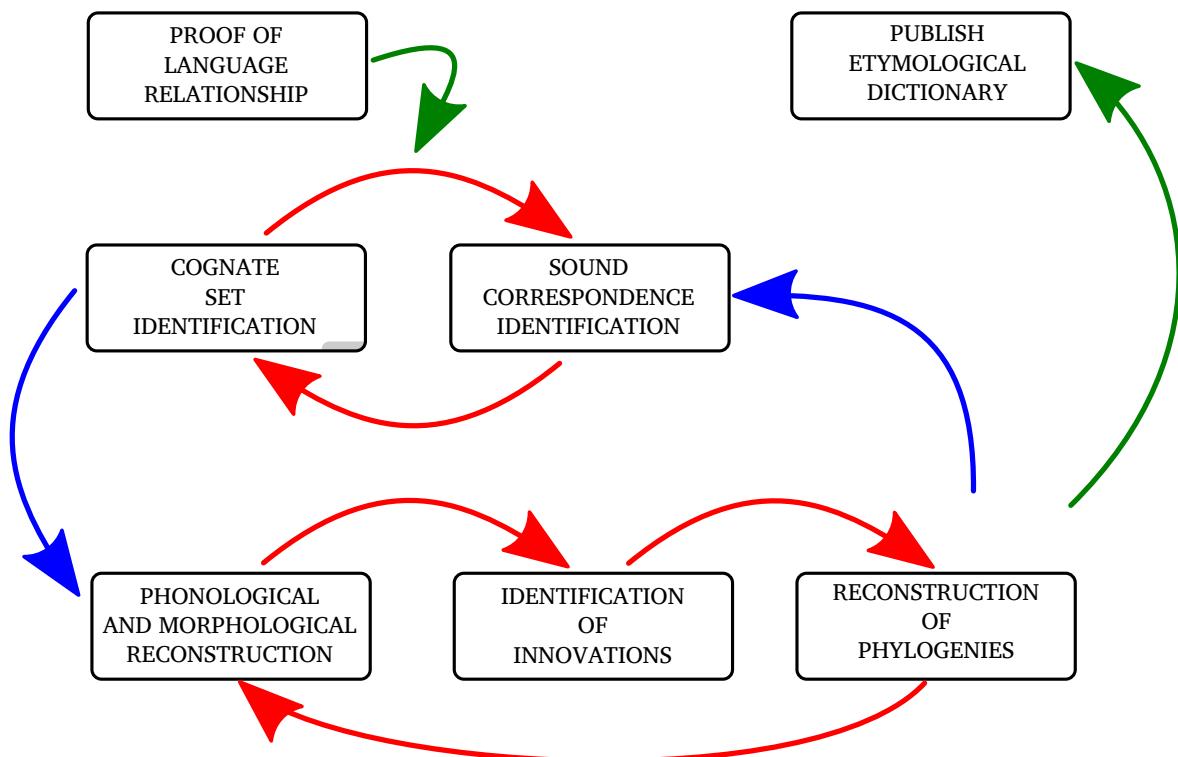
Die komparative Methode ist ein Komplex von Verfahren der historischen Sprachwissenschaft, mit deren Hilfe Sprachen klassifiziert und nicht belegte Sprachstufen rekonstruiert werden, um somit Entwicklungsgeschichte von Sprachen zu schreiben. Die Ergebnisse der komparativen Methoden werden in Form von etymologischen Wörterbüchern, historischen Grammatiken und Entwicklungsschemata (evolutionäre Bäume und Netze) kodiert.

7.3.3 Die komparative Methode

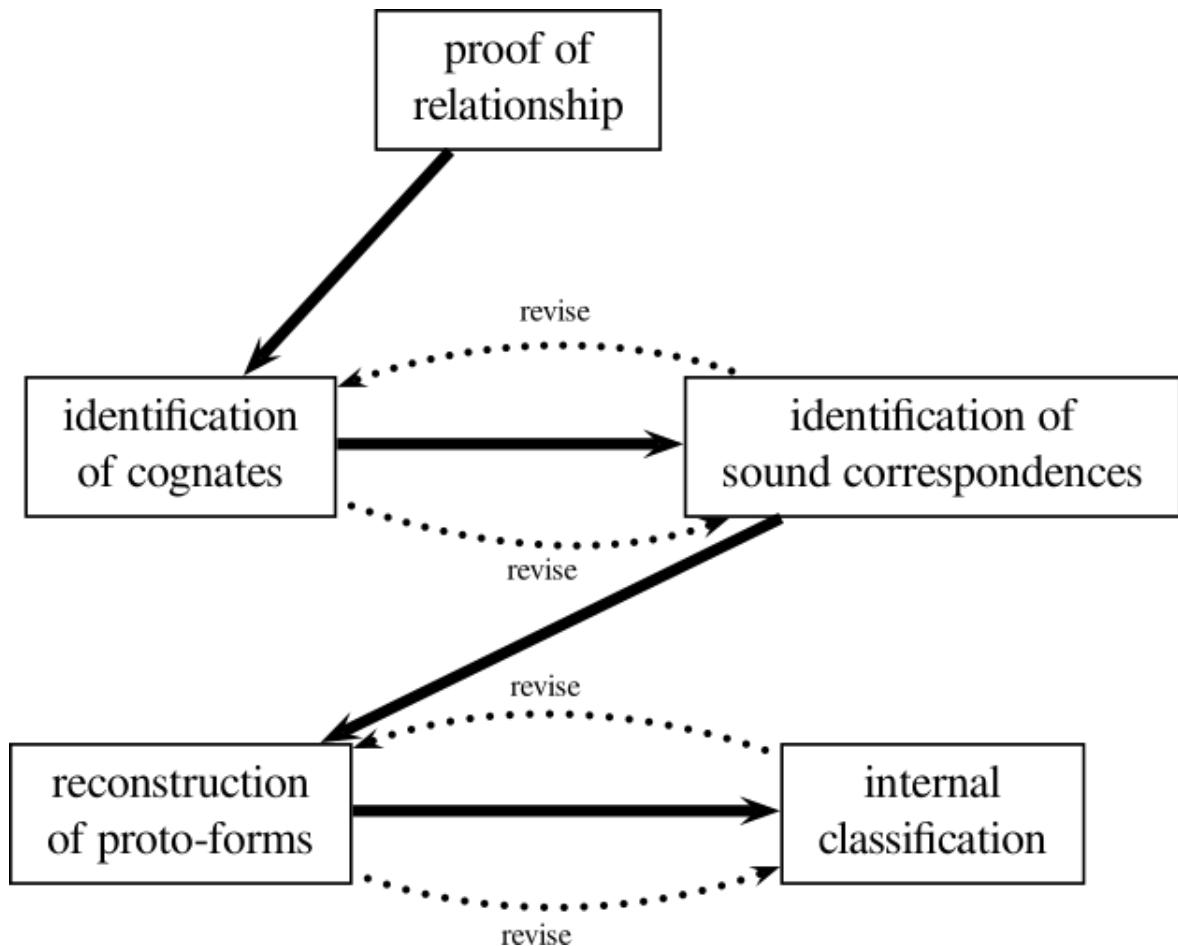
Beschreibung des Workflows in Ross und Durie (1996)

1. Determine on the strength of diagnostic evidence that a set of languages are genetically related, that is, that they constitute a ‘family’;
2. Collect putative cognate sets for the family (both morphological paradigms and lexical items).
3. Work out the sound correspondences from the cognate sets, putting ‘irregular’ cognate sets on one side;
4. Reconstruct the protolanguage of the family as follows:
 - a Reconstruct the protophonology from the sound correspondences worked out in (3), using conventional wisdom regarding the directions of sound changes.
 - b Reconstruct protomorphemes (both morphological paradigms and lexical items) from the cognate sets collected in (2), using the protophonology reconstructed in (4a).
5. Establish innovations (phonological, lexical, semantic, morphological, morpho-syntactic) shared by groups of languages within the family relative to the reconstructed protolanguage.
6. Tabulate the innovations established in (5) to arrive at an internal classification of the family, a ‘family tree’.
7. Construct an etymological dictionary, tracing borrowings, semantic change, and so forth, for the lexicon of the family (or of one language of the family).

Visualisierung des Workflows von Ross und Durie (1996)



Vereinfachter Workflow (List 2014)



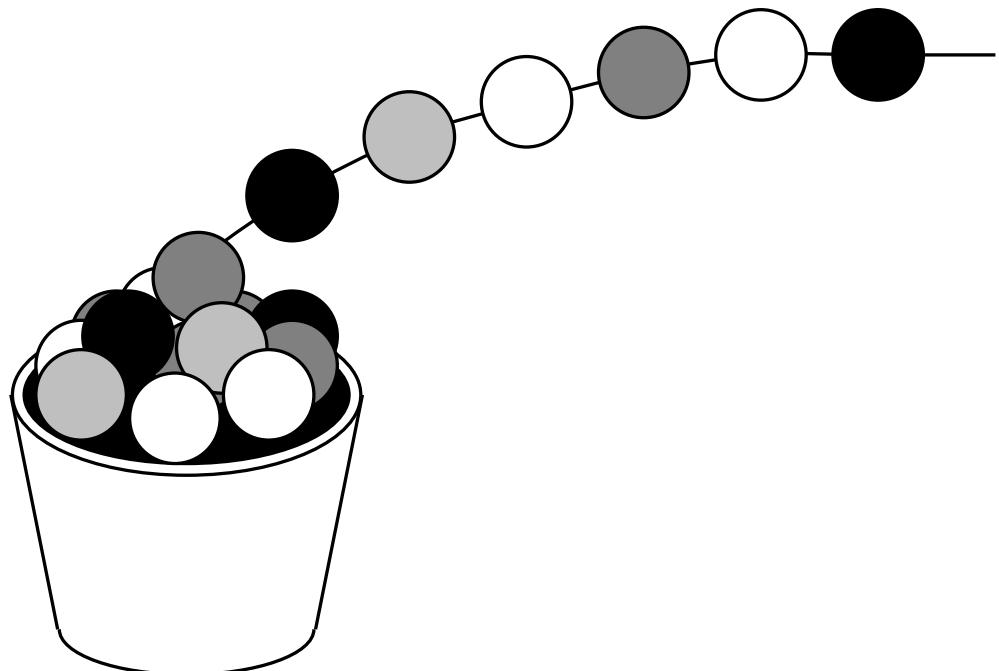
7.4 Sequenzlinierung

7.4.1 Sequenzen und Sequenzvergleiche

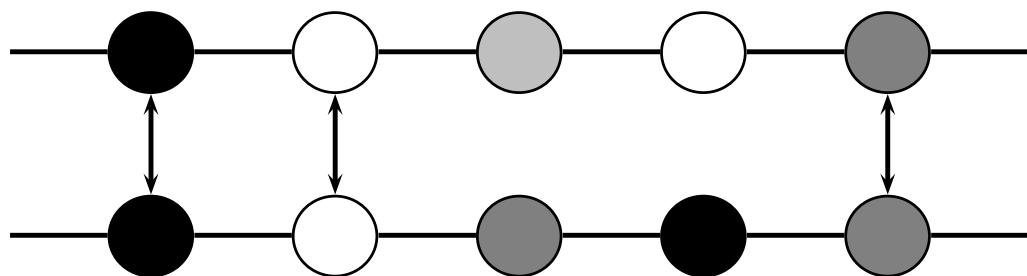
Sequenzen: Definition (vgl. Böckenhauer und Bongartz 2003: 30f)

Definition: Ein *Alphabet* ist eine nicht-leere endliche Menge deren Elemente *Buchstaben* genannt werden. Eine *Sequenz* ist eine geordnete Liste von Buchstaben, die aus dem Alphabet gezogen werden. Die Elemente von Sequenzen werden *Segmente* genannt, die *Mächtigkeit* einer Sequenz ist die Anzahl ihrer unterschiedlichen Buchstaben, und die *Länge* einer Sequenz ist die Anzahl ihrer Segmente.

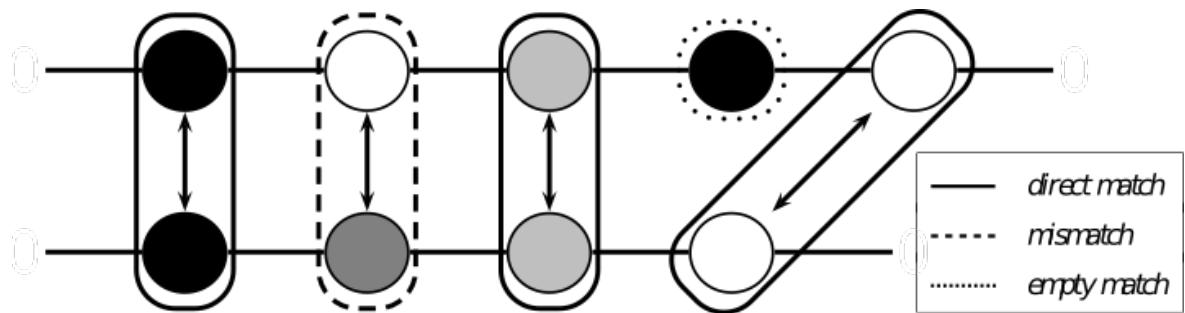
Sequenzen als Perlenketten



Vergleich von Sequenzen mit gleicher Länge



Vergleich von Sequenzen mit unterschiedlicher Länge



1

7.4.2 Alinierung

Definitionsversuch

Eine *Alinierung* von n ($n > 1$) Sequenzen ist eine n -reihige Matrix, in der alle Sequenzen dergestalt angeordnet werden, dass alle matchingen Segmente in derselben Spalte erscheinen, während nicht-matchende Segmente, die aus leeren Matches resultieren, durch Gap-Symbole angezeigt werden. (vgl Gusfield 1997: 216)

Beispiel



Preisfrage

Die Levenshtein-Distanz zwischen zwei Sequenzen S_1 und S_2 ist definiert als die Anzahl von Editieroperationen, die notwendig ist, um S_1 in S_2 zu transformieren. Mit Hilfe des Konzepts der Alinierung lässt sich dieses Distanzmaß leicht auf die Hamming-Distanz zurückführen. Wie genau?

7.4.3 Phonetische Alinierung

Obwohl Alinierungsanalysen eine der allgemeinsten Möglichkeiten darstellen, Sequenzen zu vergleichen, steckt ihre Verwendung in der historischen Linguistik noch in den Kinderschuhen. Natürlich alinieren historische Linguisten eigentlich

immer Wörter und haben dies auch schon immer getan, da ohne Alinierung überhaupt keine regulären Lautkorrespondenzen ermittelt werden könnten. Der Sprachvergleich basierte lange Zeit jedoch eher auf einer impliziten Alinierung, die selten visualisiert wurde, und wenn doch, dann nur aus illustrativen Zwecken.

Schwierigkeiten der phonetischen Alinierung

Language	Alignment						
Russian	s	-	ɔ	n	ts	ə	-
Polish	s	w	ɔ	n ^j	ts	ɛ	-
French	s	-	ɔ	l	-	ɛ	j
Italian	s	-	o	l	-	e	-
German	s	-	ɔ	n	-	ə	-
Swedish	s	-	u:	l	-	-	-

(a) Globale Alinierung

Language	Alignment						
Russian	s	ɔ	-	-	n	ts	ə
Polish	s	-	w	ɔ	n ^j	ts	ɛ
French	s	ɔ	l	-	-	-	-
Italian	s	o	l	-	-	-	e
German	s	ɔ	-	-	-	-	nə
Swedish	s	u:	l	-	-	-	-

(b) Lokale Alinierung

Substantielle und strukturelle Ähnlichkeit

- die Wörter "schlafen" und "Flaschen" sind recht ähnlich
- die Wörter "Obst" und "Post" auch
- die Wörter "Kerker" und "Tanten" sind sich auch recht ähnlich, aber auf andere Art und Weise

Die beiden unterschiedlichen Formen von Ähnlichkeit können wir "substantielle" vs. "strukturelle" Ähnlichkeit nennen.

Heuristiken für strukturelle Ähnlichkeiten

Beim Alinieren in der historischen Linguistik ist es wichtig, der Tatsache Rechnung zu tragen, dass substantielle Ähnlichkeit zwischen Lauten nicht notwendigerweise auch auf deren Kognazität hinweist. Nur, wenn zwei Segmente auch systematisch (also in den Sprachsystemen) korrespondieren, sollten sie tatsächlich als ähnlich angesehen werden. In den Schritten des Sprachvergleichs kann diese systematische Ähnlichkeit jedoch schwer ermittelt werden, denn zu Beginn eines Sprachvergleichs sind weder die kognaten Wörter bekannt, noch die regulär korrespondierenden Laute.

Dolgopolskys Lautklassen

No.	Cl.	Description	Examples
1	"P"	labial obstruents	p, b, f
2	"T"	dental obstruents	d, t, θ, ð
3	"S"	sibilants	s, z, ſ, ʒ
4	"K"	velar obstruents, dental and alveolar affricates	k, g, ts, tʃ
5	"M"	labial nasal	m
6	"N"	remaining nasals	n, ɳ, ɲ
7	"R"	liquids	r, l
8	"W"	voiced labial fricative and initial rounded vowels	v, u
9	"J"	palatal approximant	j
10	"Ø"	laryngeals and initial velar nasal	h, ɦ, ɳ

8 Sequenzalinierung in JavaScript

8.1 Automatische Sequenzalinierung

8.1.1 Das Problem

Wir wollen zwei Sequenzen automatisch alinieren, aber wie?

- Eine einfache automatische Lösung wäre es, einfach alle verschiedenen Kombinationen durchzutesten.
- In einem weiteren Schritt könnten wir dann alle Alinierungen testen und ihren "Score" berechnen, also sie bewerten, indem wir, zum Beispiel, zählen, wie viele Mismatches und leere Matches sie aufweisen.
- Dann könnten wir den Score für jedes Paar aufsummieren und würden einfach die Alinierung mit dem besten Score nehmen.

Leider dauert das zu lange!

Die Zahl aller Möglichen Alinierungen zwischen zwei Strings hängt von deren Länge ab: Es gilt:

$$N = \sum_{k=0}^{\min(m,n)} 2^k \cdot \binom{m}{k} \cdot \binom{n}{k},$$

für zwei Strings der Länge m und n .

Und was soll das heißen?

Das wir für die Alinierung von "Herz" und "heart" 681 verschiedene Alinierungen testen müssten, und für die Alinierung von "Werdegänge" und "Kinderpass" 8 097 453 verschiedene Möglichkeiten!

8.1.2 Die Lösungsstrategie

Sei faul und mach Dir keine unnötige Arbeit!

Die Lösungsstrategie beruht auf der Idee, dass man bestimmte Teillösungen, die man schon berechnet hat, ja eigentlich nicht noch ein zweites Mal berechnen muss, und ferner, dass man Lösungswege, die so abwegig sind, dass klar ist, dass sie nicht zum Erfolg führen können, einfach nicht weiterverfolgt.

Dynamische Programmierung

Der Algorithmus, mit dem man die optimale Alinierung von zwei Strings relativ schnell und effizient berechnen kann, gehört zur Familie der dynamischen Programmieralgorithmen (*dynamic programming*). Die Idee dieser Algorithmen ist es, ein Problem in umgekehrter Richtung anzugehen: Anstatt alle möglichen Alinierungen zwischen zwei Sequenzen zu testen, baut man eine Alinierung auf, indem man Gebrauch macht von “previous solutions for optimal alignments of smaller subsequenze” (Durbin et al. 1998[2002]).

8.1.3 Der Algorithmus

Grundlegende Bestandteile

1. **Scoring Schema:** bestimmt, wie wir uniforme, divergente, und leere Matches bewerten.
2. **Matrizenkonstruktion:** erstellt eine Matrix, in der wir alle möglichen Kombinationen der Sequenzen miteinander einander gegenüberstellen.
3. **Traceback:** Merkt sich alle Zwischenentscheidungen, die getroffen wurden, so dass wir nachher ermitteln können, welchen “Pfad” wir gelaufen sind, umz um besten Ergebnis zu gelangen.

Scoring Schema

Matching Type	Score	Example
uniform match	0	A / A
divergent match	1	A / B
empty match	1	A / -, - / A

Matrix

- / -	- / B	- / Ä	- / R
A / -	A / B	A / Ä	A / R
B / -	B / B	B / Ä	B / R
E / -	E / B	E / Ä	E / R
R / -	R / B	R / Ä	R / R

Traceback

- / -	- / B	- / Ä	- / R
A / -	A / B	A / Ä	A / R
B / -	B / B	B / Ä	B / R
E / -	E / B	E / Ä	E / R
R / -	R / B	R / Ä	R / R

Traceback

8.2 Sequenzalinierung in JavaScript

8.2.1 Vorüberlegungen

Um den Algorithmus zu implementieren, brauchen wir:

- einige numerische Variablen
- die Repräsentation der Sequenzen
- Wege, Alinierungen zu repräsentieren
- Wege, die Matrix zu repräsentieren
- eine Möglichkeit, das Scoring-Schema umzusetzen

JavaScript bietet uns:

- Variablen, kein Problem
- Listen (Arrays) für die Alinierungen
- und auch für die Sequenzen
- mehrdimensionale Arrays für die Matrix und den Traceback
- if/else-Verzweigungen und String-Vergleiche für das Scoring-Schema

8.2.2 Implementierung der Matrixberechnung

```
function wf_align(seqA, seqB) {
    /* Vorbereitung der Daten hier */
    /* Matrix-Initialisierung hier */
    /* Haupt-Loop hier, darin auch die Scoring Function */
    /* Traceback hier */
```

```

/* Nachbearbeitung hier */
}

function wf_align(seqA, seqB) {
    /* return nothing if either of the lists is empty */
    if(seqA.length == 0 || seqB.length == 0) {
        return;
    }
    /* get the lengths of the sequences */
    var alen = seqA.length;
    var blen = seqB.length;
    /* declare variables in local environment */
    var i, j; // numbers
    var gapA, gapB, dist; // floats
    var charA, charB; // characters
    /* Matrix-Initialisierung hier */
    /* Haupt-Loop hier, darin auch die Scoring Function */
    /* Traceback hier */
    /* Nachbearbeitung hier */
}

function wf_align(seqA, seqB) {
    /* Vorbereitung hier */
    /* create the matrix */
    var matrix = [];
    for(var i=0;i<alen+1;i++) {
        var inline = [];
        for(var j=0;j<blen+1;j++) {
            inline.push(0);
        }
        matrix.push(inline);
    }
    /* initialize matrix */
    for(i=1;i<blen+1;i++) {
        matrix[0][i] = i;
    }
    for(i=1;i<alen+1;i++) {
        matrix[i][0] = i;
    }
    /* create the traceback */
    var traceback = [];
    for(var i=0;i<alen+1;i++) {
        var inline = [];
        for(var j=0;j<blen+1;j++) {
            inline.push(0);
        }
        traceback.push(inline);
    }
    /* initialize traceback */
    for(i=1;i<blen+1;i++) {

```

```

        traceback[0][i] = 2;
    }
    for(i=1;i<alen+1;i++) {
        traceback[i][0] = 1;
    }
    /* Haupt-Loop hier, darin auch die Scoring Function */
    /* Traceback hier */
    /* Nachbearbeitung hier */
}

function wf_align(seqA, seqB) {
    /* Vorbereitung hier */
    /* Matrix-Initialisierung hier */
    /* start the iteration to fill the matrix */
    for(i=1;i<alen+1;i++) {
        for(j=1;j<blen+1;j++) {
            /* get the character in both sequences at their respective position */
            charA = seqA[i-1];
            charB = seqB[j-1];
            /* check the similarity between the characters to get the local distance */
            if(charA == charB) {
                dist = matrix[i-1][j-1];
            }
            else {
                dist = matrix[i-1][j-1]+1;
            }
            /* we have the distance for substitution, now we need the gaps */
            gapA = matrix[i-1][j]+1;
            gapB = matrix[i][j-1]+1;
            /* find the minimal value */
            if(dist < gapA && dist < gapB) {
                matrix[i][j] = dist;
            }
            else if(gapA < gapB) {
                matrix[i][j] = gapA ;
                traceback[i][j] = 1;
            }
            else {
                matrix[i][j] = gapB;
                traceback[i][j] = 2;
            }
        }
    }
    /* Traceback hier */
    /* Nachbearbeitung hier */
}

```

8.2.3 Implementierung des Traceback

```
function wf_align(seqA, seqB) {
```

```

/* Vorbereitung hier */
/* Matrix-Initialisierung hier */
/* get indices for the last cells of the matrix */
i = matrix.length-1;
j = matrix[0].length-1;
/* get the edit-dist */
var ED = matrix[i][j];
/* initialize the alignment arrays */
var almA = [];
var almB = [];
/* start the traceback */
while(i > 0 || j > 0) {
    if(traceback[i][j] == 0) {
        almA.push(seqA[i-1]);
        almB.push(seqB[j-1]);
        i--;
        j--;
    }
    else if(traceback[i][j] == 1) {
        almA.push(seqA[i-1]);
        almB.push("-");
        i--;
    }
    else {
        almA.push("-");
        almB.push(seqB[j-1]);
        j--;
    }
}
/* Nachbearbeitung hier */
}

function wf_align(seqA, seqB) {
    /* Vorbereitung hier */
    /* Matrix-Initialisierung hier */
    /* Haupt-Loop hier, darin auch die Scoring Function */
    /* reverse alignments */
    almA = almA.reverse();
    almB = almB.reverse();
}

```

8.3 Interaktive Sequenzalinierung

8.3.1 Grundlagen

Input-Felder in HTML

Um eine interaktive Alinierungsapp zu erstellen, machen wir Gebrauch von den Input-Felder in HTML. Diese sind sehr einfach zu erstellen:

Beachten Sie dabei, dass wir unbedingt eine ID als Tag vergeben müssen, wie auch eine Funktion, die wir diesmal nicht per `onclick="function()"`, sondern mittels `onkeyup="function()"` ausführbar machen. Das heißt nichts anderes, als dass jedes mal, wenn wir eine Taste drücken, während wir mit dem Cursor im Text-Feld sind, potentiell eine Funktion auslösen.

Es kann losgehen!

Abgesehen davon brauchen wir eigentlich nur noch die klassische Einbindung von JavaScript in eine HTML-Seite vorzunehmen. Wir nutzen dafür das Script nw.js, welches zuvor besprochen wurde. Auch eine kleine CSS-Datei wird erstellt, damit es nachher ein wenig besser aussieht. Darauf wird in diesem Zusammenhang aus Zeitgründen nicht genauer eingegangen), die findet sich aber online und auf GitHub und kann dort gezielt eingesehen werden.

8.3.2 Implementierung

Der Header

```
<html>
<head>
  <title>Alignment Demo</title>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <script src="js/nw.js" type="text/javascript"></script>
  <link rel="stylesheet" type="text/css" href="css/nw.css" />
</head>
```

Der Body

```
<body>
  <input type="text" style="width: 300px" id="seqA" onkeyup="alignit()" />
  <input type="text" style="width: 300px" id="seqB" onkeyup="alignit()" />
  <div id="display"></div>
<!-- Java-Script-Code kommt hier --&gt;
&lt;/body&gt;</pre>
```

Der JavaScript-Code

```
function alignit() {
  /* get the sequences */
  var seqA = document.getElementById('seqA');
  var seqB = document.getElementById('seqB');
```

```

/* get the values */
seqA = seqA.value.split('');
seqB = seqB.value.split('');
/* return if one of the vals is none */
if (seqA == '' || seqB == '') {
    document.getElementById('display').innerHTML = '';
    return;
}
/* get the alignment */
alms = wf_align(seqA, seqB);
var almA = alms[0];
var almB = alms[1];
var dist = alms[2];
/* create the text */
var txt = '';
txt += ''+almA.join('')+';
txt += ''+almB.join('')+';
txt += 'Edit-Distance: '+dist+'';
txt += '';
/* update */
document.getElementById('display').innerHTML = txt;
}

```

8.3.3 Demo

9 Sequenzalinierung in Python

9.1 Sequenzalinierung in Python

9.1.1 Allgemeine Strategien

Python und JavaScript

Python und JavaScript sind sich im Prinzip recht ähnlich, was die Datentypen anbelangt. Wir werden auch für unsere Python-Implementierung auf Listen (Arrays in JavaScript) zurückgreifen, und Loops durchführen (**while** für den Traceback und **for** für die Matritzen-Füllung). Allerdings können wir in Python kompakter formulieren und haben es daher etwas leichter, den Kode aufzusetzen.

Struktur

Die Implementierung des Alinierungsalgoritmus in Python (eine Version des Wagner-Fischer Algorithmus) basiert auf den gleichen Blöcken, wie auch die Alinierung in der JavaScript-Implementierung, die wir in der vorigen Sitzung besprochen haben:

1. Vorbereitung von Konstanten und allgemeine Checks
2. Initialisierung von Matrix und Traceback
3. Ausfüllen von Matrix und Traceback

4. Traceback und Konstruktion der Alinierungen

Dann kann's ja losgehen!

Genau! Denn der Algorithmus ist tatsächlich sehr leicht in Python zu implementieren, vorausgesetzt man ist mit den Datentypen ein wenig vertraut. Aber wir schauen uns das jetzt Schritt für Schritt in Ruhe an.

9.1.2 Implementierung der Matrixberechnung

Grundlegender Aufbau

```
def wf_align(seqA, seqB):
    """
    BLABLA
    """

    # Vorgeplänkel
    # Initialisierung von Matrix und Traceback
    # Ausfüllen von Matrix und Traceback
    # Traceback-Vorbereitung
    # Traceback
    # Traceback-Nachbereitung

    return almA, almB, ED
```

Vorgeplänkel

```
def wf_align(seqA, seqB):
    """
    Align two sequences using the Wagner-Fisher algorithm.
    """

    # check for empty seqs
    if not seqA or not seqB:
        return

    # store length of sequences
    m = len(seqA)+1
    n = len(seqB)+1

    # Initialisierung von Matrix und Traceback
    # Ausfüllen von Matrix und Traceback
    # Traceback-Vorbereitung
```

```

# Traceback
# Traceback-Nachbereitung

return almA, almB, ED

```

Initialisierung von Matrix und Traceback

```

def wf_align(seqA, seqB):
    """
    Align two sequences using the Wagner-Fisher algorithm.
    """

    # Vorgeplänkel
    # Initialisierung von Matrix und Traceback
    M = [[0 for i in range(n)] for j in range(m)]
    T = [[0 for i in range(n)] for j in range(m)]

    # initialize M and T
    for i in range(m):
        M[i][0] = i
    for i in range(n):
        M[0][i] = i
    for i in range(1,m):
        T[i][0] = 1
    for i in range(1,n):
        T[0][i] = 2

    # Ausfüllen von Matrix und Traceback
    # Traceback-Vorbereitung
    # Traceback
    # Traceback-Nachbereitung

    return almA, almB, ED

```

Ausfüllen von Matrix und Traceback

```

def wf_align(seqA, seqB):
    """
    Align two sequences using the Wagner-Fisher algorithm.
    """

    # Vorgeplänkel
    # Initialisierung von Matrix und Traceback
    # Ausfüllen von Matrix und Traceback

    # start the main loop
    for i in range(1,m):
        for j in range(1,n):

```

```

# get the chars
charA = seqA[i-1]
charB = seqB[j-1]

# check identity
if charA == charB:
    match = M[i-1][j-1]
else:
    match = M[i-1][j-1] + 1

# get the gaps
gapA = M[i-1][j] + 1
gapB = M[i][j-1] + 1

# compare the stuff
if match <= gapA and match <= gapB:
    M[i][j] = match
elif gapA <= gapB:
    M[i][j] = gapA
    T[i][j] = 1 # don't forget the traceback
else:
    M[i][j] = gapB
    T[i][j] = 2 # don't forget the traceback

# Traceback-Vorbereitung
# Traceback
# Traceback-Nachbereitung

return almA, almB, ED

```

9.1.3 Implementierung des Traceback

Traceback-Vorbereitung

```

def wf_align(seqA, seqB):
    """
    Align two sequences using the Wagner-Fisher algorithm.
    """

    # Vorgeplänkel
    # Initialisierung von Matrix und Traceback
    # Ausfüllen von Matrix und Traceback
    # Traceback-Vorbereitung
    # get the edit distance
    ED = M[i][j]

    # start the traceback
    i,j = m-1,n-1

```

```

almA, almB = [], []
# Traceback
# Traceback-Nachbereitung
return almA, almB, ED

```

Traceback

```

def wf_align(seqA, seqB):
    """
    Align two sequences using the Wagner-Fisher algorithm.
    """
    # Vorgeplänkel
    # Initialisierung von Matrix und Traceback
    # Ausfüllen von Matrix und Traceback
    # Traceback-Vorbereitung
    # Traceback
    while i > 0 or j > 0:
        if T[i][j] == 0:
            almA += [seqA[i-1]]
            almB += [seqB[j-1]]
            i -= 1
            j -= 1
        elif T[i][j] == 1:
            almA += [seqA[i-1]]
            almB += ["-"]
            i -= 1
        else:
            almA += ["-"]
            almB += [seqB[j-1]]
            j -= 1
    # Traceback-Nachbereitung
    return almA, almB, ED

```

Traceback

```

def wf_align(seqA, seqB):
    """
    Align two sequences using the Wagner-Fisher algorithm.
    """
    # Vorgeplänkel
    # Initialisierung von Matrix und Traceback

```

```

# Ausfüllen von Matrix und Traceback
# Traceback-Vorbereitung
# Traceback
# Traceback-Nachbereitung

# reverse
almA = almA[::-1]
almB = almB[::-1]

return almA, almB, ED

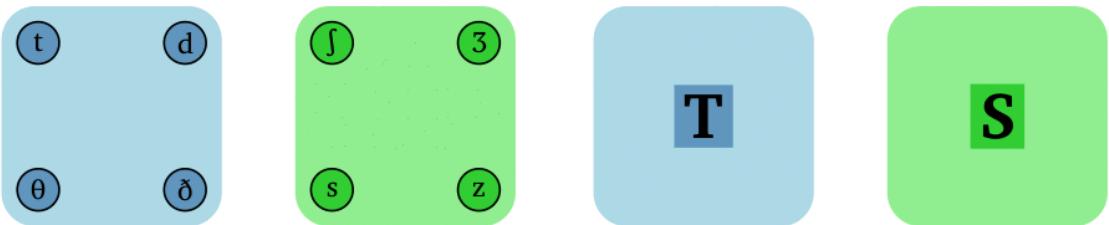
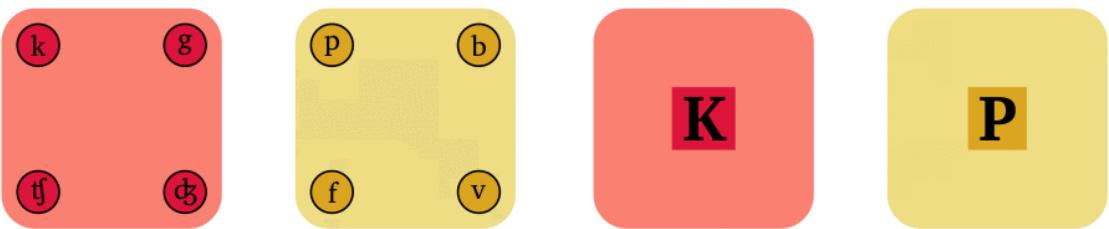
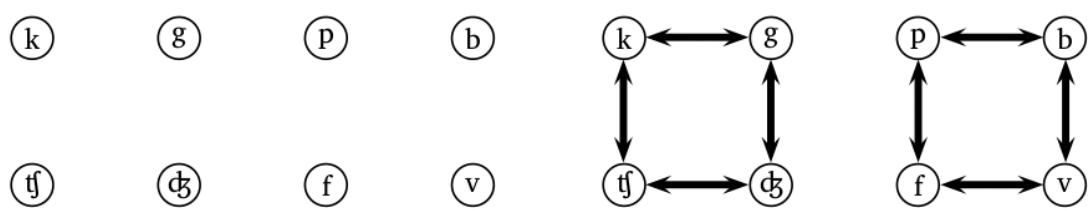
```

9.2 Lautklassenbasierte Alinierung

9.2.1 Noch mal zu den Lautklassen

Dolgopolskys Idee

Sounds which frequently occur in correspondence relations in genetically related languages can be clustered into classes. It is thereby assumed that “phonetic correspondences inside a ‘type’ are more regular than those between different ‘types’ ” (Dolgopolsky 1986[1964]:35).



Dolgopolskys Lautklassenmodell

No.	Cl.	Description	Examples
1	"P"	labial obstruents	p, b, f
2	"T"	dental obstruents	d, t, θ, ð
3	"S"	sibilants	s, z, ſ, ʒ
4	"K"	velar obstruents, dental and alveolar affricates	k, g, ts, tʃ
5	"M"	labial nasal	m
6	"N"	remaining nasals	n, ŋ, ɳ
7	"R"	liquids	r, l
8	"W"	voiced labial fricative and initial rounded vowels	v, u
9	"J"	palatal approximant	j
10	"Ø"	laryngeals and initial velar nasal	h, fi, ɳ

Erweitertes Lautklassenmodell (List 2014)

No.	Cl.	Description	Examples
1	"A"	unrounded back vowels	a, ɑ
2	"B"	labial fricatives	f, β
3	"C"	dental / alveolar affricates	ts, dʐ, tʃ, ɖ
4	"D"	dental fricatives	θ, ð
5	"E"	unrounded mid vowels	e, ε
6	"G"	velar and uvular fricatives	ɣ, x
7	"H"	laryngeals	h, ?
8	"I"	unrounded close vowels	i, ɪ
9	"J"	palatal approximant	j
10	"K"	velar and uvular plosives	k, g
11	"L"	lateral approximants	l
12	"M"	labial nasal	m
13	"N"	nasals	n, ɳ
14	"O"	rounded back vowels	œ, ɒ
15	"P"	labial plosives	p, b
16	"R"	trills, taps, flaps	r
17	"S"	sibilant fricatives	s, z, ʃ, ʒ
18	"T"	dental / alveolar plosives	t, d
19	"U"	rounded mid vowels	ɔ, o
20	"W"	labial approx. / fricative	v, w
21	"Y"	rounded front vowels	u, ʊ, y
22	"0"	low even tones	11, 22
23	"1"	rising tones	13, 35
24	"2"	falling tones	51, 53
25	"3"	mid even tones	33
26	"4"	high even tones	44, 55
27	"5"	short tones	1, 2
28	"6"	complex tones	214

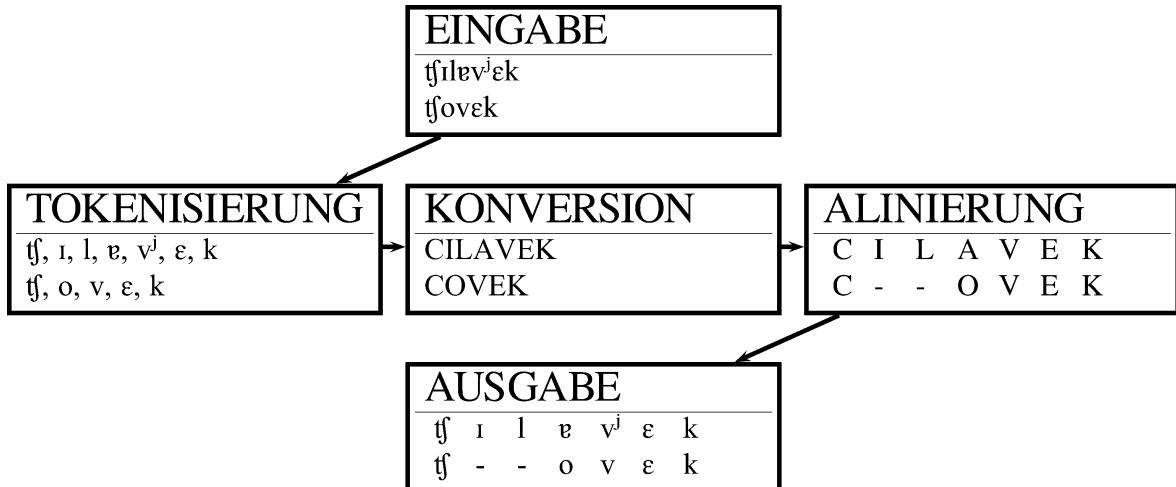
9.2.2 Lautklassenkonvertierung mit Python

Grundlegende Idee

- Um eine Sequenz in Lautklassen zu konvertieren, müssen wir für jeden möglichen Lautbuchstaben den entsprechenden Lautklassenbuchstaben definieren.
- Am besten dafür geeignet ist ein Python-Dictionary, welches aus Key-Value-Paaren besteht und zu jedem Key den Value liefern kann.
- Das Erstellen eines solchen Dictionaries ist zwar nervig, es ist aber die einfachste Möglichkeit, um nachher ein Wort zu konvertieren.

- Das Wort muss allerdings segmentiert vorliegen, da wir ansonsten keine Möglichkeit haben, zu wissen, was jeweils ein Laut sein soll (vgl. Laute wie [t^h] oder [ts], die aus zwei oder mehr Zeichen bestehen).

Workflow



Segmentierung

Für die Segmentierung bedienen wir uns eines einfachen Modells: Jede Sequenz wird mit Hilfe des Leerzeichens segmentiert dargestellt:

- Tochter*: "t^hɔxt^hər" → "t^h ɔ x t^h ə r"
- daughter*: "dɔ:t^hər" → "d ɔ: t^h ə r"

Mit Python können wir diesen Input ganz schnell in eine Liste umwandeln:

```
>>> my_string = "th ɔ x th ə r"
>>> my_sequence = my_string.split(" ")
>>> my_sequence
['th', 'ɔ', 'x', 'th', 'ə', 'r']
```

Umwandlung

Bei der Umwandlung mit Hilfe eines Dictionaries sollten wir damit rechnen, dass bestimmte Zeichen nicht vorhanden sind (es gibt so viele Symbole, und es kann fast unmöglich sein, alle zu finden, außerdem können die Daten auch Fehler enthalten). Daher führen wir einen dreistufigen Test ein.

```
def segment2class(segment, converter):
    """
    Convert a segment to a sound-class schema.
    """
    # erster versuch
    try:
```

```

        return converter[segment]
    except KeyError:
        # zweiter versuch
        try:
            return converter[segment[0]]
        except KeyError:
            # ansonsten, gib den "misserfolg"-character zurück
            return '0'

```

9.2.3 Implementierung

Laden des Lautklassenmodells

Wir speichern und laden den Converter (dolgo.json) im json-Format, welches zu einem der gängigsten Austauschformate geworden ist. Wir laden die Datei mit Hilfe des **json**-Modules, wo uns automatisch ein Python-Dictionary zurückgegeben wird.

```

def load_model(model):
    """
    Load the converter for a sound-class model.
    """
    # load the converter with json
    converter = json.load(open(model+'.json'))
    return converter

```

Lautklassenkonvertierung

Die Konvertierung lässt sich mit sehr wenigen Zeilen Kode umsetzen, da wir hier von der Listengenerationssyntax in Python Gebrauch machen können (so genannte list comprehension):

```

def segments2classes(segments, converter):
    """
    Convert a sound string to a sound-class string.
    """
    # check for segmented string
    if type(segments) == str:
        segments = segments.split(' ')
    # convert the segments
    classes = [segment2class(x) for x in segments]
    return classes

```

Rückkonvertierung

Zur Rückkonvertierung brauchen wir den ursprünglichen String. Wir gehen ja davon aus, dass wir die Strings zuerst in Lautklassen umwandeln, diese dann

alinieren, und nachher alles zurückkonvertieren. Hierbei müssen wir im Prinzip nur merken, wo die Gaps eingesetzt wurden, wir müssen also die Gaps aus der Lautklassensequenz in die ursprüngliche Sequenz übertragen:

```
def classes2segments(classes, segments):
    """
    Convert an aligned string of sound classes back to the string of segments.
    """
    idx = len(segments)-1
    out = []
    for i in range(len(classes)-1,-1,-1):
        print(i,classes[i])
        if classes[i] == '-':
            out += [classes[i]]
        else:
            out += [segments[idx]]
            idx -= 1
    return out[::-1]
```

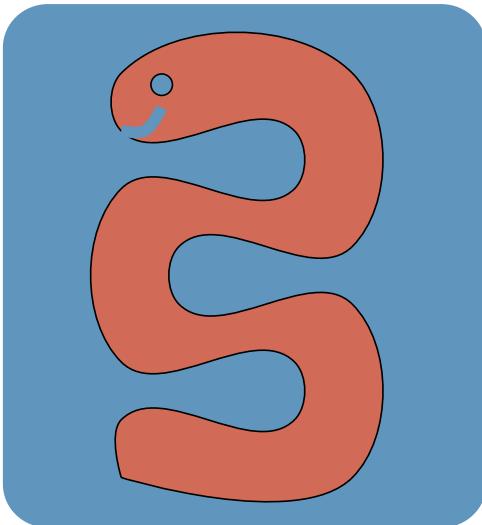
Vollständige Lautklassenbasierte Alinierung

```
def sca_align(stringA, stringB, model="dolgo"):
    """
    Carry out sound-class based alignment analysis.
    """
    # check for strings passed as such
    if type(stringA) == str:
        stringA = stringA.split(' ')
        stringB = stringB.split(' ')
    # load the converter
    converter = load_model(model)
    # Konvertierung
    seqA = segments2classes(stringA, converter)
    seqB = segments2classes(stringB, converter)
    # Alinierung
    almA, almB, ED = wf_align(seqA, seqB)
    # Rück-Konvertierung
    outA = classes2segments(almA, stringA)
    outB = classes2segments(almB, stringB)
    return outA, outB, ED
```

9.3 Phonetische Alinierung mit LingPy

9.3.1 Was ist LingPy?

Grundlegendes zu LingPy



Software Library for Automatic Tasks in Historical Linguistics

- phonetic segmentation
- phonetic alignment
- cognate detection
- ancestral state reconstruction
- borrowing detection
- phylogenetic reconstruction

GitHub This repository Search Explore Features Enterprise Blog Sign up Sign in

lingpy / lingpy Watch 11 Star 14 Fork 10

LingPy: Python library for quantitative tasks in historical linguistics <http://lingpy.org>

1,014 commits 2 branches 6 releases 11 contributors

branch: master +

modified release in setup.py	latest commit 8214242a76
LinguList authored 4 days ago	
renaming project-level directory used for some development tests	8 months ago
fixes #136	7 months ago
updated added collexification menu, initial ideas	4 months ago
Merge branch 'master' of https://github.com/LinguList/lingpy	7 months ago
added tests for phylogeny.py and refactored the module to unify file ...	7 months ago
Merge branch 'master' of https://github.com/LinguList/lingpy	7 months ago
fix markup	8 months ago

Code Issues 17 Pull requests 0 Pulse Graphs

HTTPS clone URL <https://github.com/LinguList/lingpy> You can clone with [HTTPS](#) or [Subversion](#).

Download ZIP

9.3.2 Grundlegende Befehle

Arbeiten mit Sequenzen

```
>>> from lingpy import *
>>> seq = sampa2uni('t_h0xt_h@r')
>>> seq
'tʰɔxtʰər'
>>> tokens = ipa2tokens(seq)
>>> tokens
['tʰ', 'o', 'x', 't', 'ə', 'r']
>>> classes = tokens2class(tokens, 'sca')
>>> classes
['T', 'U', 'G', 'T', 'E', 'R']
```

```

>>> classes = tokens2class(tokens, 'dolgo')
>>> classes
['T', 'V', 'K', 'T', 'V', 'R']
>>> from lingpy.sequence.sound_classes import *
>>> syllabify(seq)
['tʰ', 'ɔ', 'x', '◦', 'tʰ', 'ə', 'r']

```

Vergleichen von Sequenzen

```

>>> seq1 = ipa2tokens(sampa2uni('t_h0xt_h@r'))
>>> seq2 = ipa2tokens('dɔ:tʰər')
>>> seq3 = ipa2tokens('dɔttər')
>>> edit_dist(seq1, seq2)
3
>>> nw_align(seq1,seq2)
([['tʰ', 'ɔ', 'x', 'tʰ', 'ə', 'r'], ['- ', 'd', 'ɔ:', 'tʰ', 'ə', 'r'], 0.0)
>>> sw_align(seq1, seq2)
(([['tʰ', 'ɔ', 'x'], ['tʰ', 'ə', 'r'], []], ([['d', 'ɔ:'], ['tʰ', 'ə', 'r'], []]),
>>> we_align(seq1,seq2)
([[['tʰ', 'ə', 'r'], ['tʰ', 'ə', 'r'], 3.0])
>>> mult_align([seq1,seq2,seq3], pprint=True)
tʰ ɔ x tʰ ə r
d ɔ: - tʰ ə r
d ɔ - tt ə r

```

9.3.3 Workflows

Grundlegendes vorweg

Mit LingPy lassen sich gezielt Daten im großen Stil analysieren. Dafür braucht man einen Workflow, und man schreibt dann normalerweise ein kleines Skript, mit dessen Hilfe sich die Berechnungen (zum Beispiel die Alinierung von 100 verschiedenen Wortlisten) in einem "Rutsch" erledigen lassen.

In dieser Sitzung werde ich nur einen kleinen Teil eines Workflows vorstellen, und zwar die Möglichkeit, eine Alinierung automatisch vorzunehmen.

Das Input-Format

Das Input-Format für Alinierungen in Textdateien ist relativ einfach:

- die erste Zeile enthält einen Verweis auf den Datensatz
- die zweite Zeile enthält einen Verweis auf die Gruppe von Wörter, die aliniert werden sollen
- die folgenden Zeilen enthalten nun, separiert durch einen Tabstop, den Namen der Sprache und die Daten in IPA.
- wer seine Daten vorsegmentieren will, kann das tun und einfach einen String durch Leertasten separieren, alternativ kann man das auch von LingPy automatisch übernehmen lassen

Das Input-Format: Beispiel

French

French *le chasseur* (< Latin *captiatoře*), Chart 447
Chevroux..... ts a ſ a:
Vaugondry..... ts a ſ ø
L'Auberson..... tʃ a s ø
Vallorbe..... tʃ a ſ o:
Le Sentier..... ts a ſ au
Longirod..... ts a f j a:
Commugny..... θ ε f j œi
Vullierens..... tʃ a s a:
Arnex..... tʃ a ſ au
Villars-le-Terroir ts a ſ a:
Prahins..... ts a ſ ɔ:
Montpreveyres..... ts a ſ a:
Charnex..... ts e ç æ̞
Roche..... ts a ç ø:
Ormont-Dessus..... ts a t̪ au
Château-d'Œx..... ts a ç au
St-Gingolph..... ts a f j au
Collombey..... ts a f j œi
Champéry..... ts a ç œ:
Martigny..... ts a f j œi
Orsières..... ts a s j ɜu
Lourtier..... ts a ç ɔ:
Fully..... ts a ç œu
Conthey..... ts a θ j œi
Nendaz..... tʃ a ſ ç ɜu
Savièse..... ts a s j u
Ayent..... ts a ç øi
Miège..... ts a s j ou
Grône..... ts a ſ ç ou k
Évolène..... ts a ſ j ou
Grimentz..... ts a s j ou
Collex..... θ e f j ø
Vernier..... θ ε f j ø
Laconnex..... θ ε f j œ
Veyrier..... θ ε f j ø
Hermance..... θ ε f j œ
Semsalves..... ts a ç a
Montbovon..... ts a ç a:
Arconciel..... ts a ç a:
Avry-sur-Matran... ts a ç j a:
Courtepin..... ts a ç a:
Dompierre..... ts a ç a:
Murist..... ts a ç a:
Sugiez..... tʃ æ ſ ɔ
Montalchez..... ts a ſ a

Boudry.....	ts a s œ: r
Corcelles.....	ʃ a s y
Landeron.....	ʃ a s e: r
Savagnier.....	ʃ a s ø: r
Côte-aux-Fées.....	ts a ʃ œ
Noiraigue.....	ʃ a ʃ œ
Chaux-du-Milieu...	ʃ a s y
Cerneux-Péquignot.	ʃ ε s u
Lamboing.....	ʃ a s œ
Orvin.....	ʃ s u
Plagne.....	ʃ s u
Sombeval.....	ʃ s u
Court.....	ʃ s u
Vermes.....	ʃ s u
Develier.....	ʃ ə s u
Cerlatez.....	ʃ æ s u
Courtedoux.....	ʃ s u

Der Alinierungsworkflow: Multiple Alinierung

```
>>> from lingpy import *
>>> msa = Multiple('data/french-chasseur(msq')
>>> msa.prog_align()
>>> print(msa)
ts a ʃ - œ: -
ts a ʃ - ø -
ʃ a s - ø -
ʃ a ʃ - o: -
ts a ʃ - au -
ts a f j a: -
θ ε f j œi -
ʃ a s - œ: -
ʃ a ʃ - au -
ts a ʃ - œ: -
ts a ʃ - œ: -
ts a ʃ - œ: -
ts e ç - æœ -
ts a ç - ø: -
ts a t - au -
ts a ç - au -
ts a f j au -
ts a f j œi -
ts a ç - œ: -
ts a f j œi -
ts a s j ɔu -
ts a ç - œ: -
ts a ç - œu -
ts a θ j œi -
```

ſ	a	ʃ	ç	ɜu	-
ts	a	s	j	u	-
ts	a	ç	-	øi	-
ts	a	s	j	ou	-
ts	a	ʃ	ç	ou	k
ts	a	ʃ	j	ou	-
ts	a	s	j	ou	-
θ	e	f	j	ø	-
θ	ɛ	f	j	ø	-
θ	ε	f	j	œ	-
θ	ɛ	f	j	ø	-
θ	ɛ	f	j	œ	-
ts	a	ç	-	a	-
ts	a	ç	-	a:	-
ts	a	ç	-	a:	-
ts	a	ç	j	a:	-
ts	a	ç	-	a:	-
ts	a	ç	-	a:	-
ts	a	ç	-	a:	-
ſ	æ	ʃ	-	ɔ	-
ts	a	ʃ	-	a	-
ts	a	s	-	æ:	r
ſ	a	s	-	y	-
ſ	a	s	-	e:	r
ſ	a	s	-	ø:	r
ts	a	ʃ	-	œ	-
ſ	a	ʃ	-	œ	-
ſ	a	s	-	y	-
ſ	ɛ	s	-	u	-
ſ	a	s	-	œ	-
ſ	-	s	-	u	-
ſ	-	s	-	u	-
ſ	-	s	-	u	-
ſ	-	s	-	u	-
ſ	ə	s	-	u	-
ſ	æ	s	-	u	-
ſ	-	s	-	u	-

Der Alinierungsworkflow: Output

```
>>> msa.output('msa', filename='data/french-chasseur')
>>> msa.output('html', filename='data/french-chasseur')
>>> msa.output('tex', filename='data/french-chasseur')
```

Der Alinierungsworkflow: Ergebnisse (HTML-Output)

Der Alinierungsworkflow: Ergebnisse (LaTeX-Output)

```
\documentclass[xetex,12pt]{scrartcl}
\usepackage{xltextra,polyglossia,color,colortbl}
\usepackage{pspicture,pst-node,xcolor}
\usepackage{geometry}
\usepackage{array}

\setmainfont[Mapping=tex-text,Scale=1.0]{Charis SIL}
\setsansfont[Mapping=tex-text,Scale=1.0]{FreeSans}
\setmonofont{DejaVu Sans Mono}

\setmainlanguage{english}

% set the new font width, declare the new lengths first
\newlength\MyWidth
\newlength\MyHeight
% set the new lengths
\setlength\MyWidth{13.50000cm} % hier die Breite eintragen
\setlength\MyHeight{32.00000cm} % hier die Höhe eintragen

% set new lengths to put the picture
\newlength\MyWput
\newlength\MyHput
\setlength\MyWput{\MyWidth}
\setlength\MyHput{\MyHeight}
% calculate the values by dividing etc.
\divide\MyWput by 2
\divide\MyHput by 2
\addtolength\MyHput{-\MyHeight}
\addtolength\MyHput{10 pt}
\makeatletter
\addtolength\MyHput{\@ptsize pt}
\makeatother

\geometry{papersize={\MyWidth,\MyHeight},total={\MyWidth,\MyHeight}}


\parindent 0pt
\begin{document}\thispagestyle{empty}

% define the colors
%\definecolor{colV}{HTML}{C86464}
%\definecolor{colK}{HTML}{C89664}
%\definecolor{colP}{HTML}{C8C864}
%\definecolor{colH}{HTML}{96C864}
%\definecolor{colJ}{HTML}{64C864}
%\definecolor{colM}{HTML}{64C896}
%\definecolor{colN}{HTML}{64C8C8}
%\definecolor{colS}{HTML}{6496C8}
```

```
%\definecolor{colR}{HTML}{6464C8}
%\definecolor{colT}{HTML}{9664C8}
%\definecolor{colW}{HTML}{C864C8}
%\definecolor{col1}{HTML}{C86496}
%\definecolor{col2}{HTML}{BBBBBB}
%\definecolor{colX}{HTML}{C0C0C0}

\definecolor{colV}{HTML}{808080}
\definecolor{colK}{HTML}{F2F2F2}
\definecolor{colP}{HTML}{E6E6E6}
\definecolor{colH}{HTML}{B2B2B2}
\definecolor{colJ}{HTML}{999999}
\definecolor{colM}{HTML}{BFBFBF}
\definecolor{colN}{HTML}{BFBFBF}
\definecolor{cols}{HTML}{CCCCCC}
\definecolor{colR}{HTML}{A6A6A6}
\definecolor{colT}{HTML}{D9D9D9}
\definecolor{colW}{HTML}{8C8C8C}
\definecolor{col1}{HTML}{FFFFFF}
\definecolor{col2}{HTML}{737373}
\definecolor{colX}{HTML}{FFFFFF}

\rput(\MyWput,\MyHput){%
\pspicture[showgrid=false](0,0)(\the\MyWidth,\the\MyHeight)
% Insert your Code here, as you like! Coordinate system starts with 0,0 on the left page.
\rput(6.75,15.75){%
\tabular{lcccccc}
\bf\ttfamily Taxon & \multicolumn{6}{l}{\bf\ttfamily Alignment} \\
\ttfamily Chevroux&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Vaugondry&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily L'Auberson&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Vallorbe&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Le Sentier&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Longirod&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colP}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Commugny&\cellcolor{colT}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colP}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Vullierens&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Arnex&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Villars-le-Terroir&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Prahins&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Montpreveyres&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Charnex&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Roche&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Ormont-Dessus&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colR}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Château-d'Oex&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily St-Gingolph&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colP}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Collombey&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colP}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Champéry&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colS}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}\\
\ttfamily Martigny&\cellcolor{colK}\&\cellcolor{colV}\&\cellcolor{cols}\&\cellcolor{colP}\&\cellcolor{colL}\&\cellcolor{colM}\&\cellcolor{colN}
```

```

\ttfamily Orsières&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}s&\cellco
\ttfamily Lourtier&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}ç&\cellco
\ttfamily Fully&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}ç&\cellcolor
\ttfamily Conthey&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colT}ø&\cellco
\ttfamily Nendaz&\cellcolor{colK}f&\cellcolor{colV}a&\cellcolor{colS}f&\cellcolor
\ttfamily Savièse&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}s&\cellco
\ttfamily Ayent&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}ç&\cellcolor
\ttfamily Miège&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}s&\cellcolor
\ttfamily Grône&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}f&\cellcolor
\ttfamily Évolène&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}f&\cellcolor
\ttfamily Grimentz&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}s&\cellcolor
\ttfamily Collex&\cellcolor{colT}ø&\cellcolor{colV}e&\cellcolor{colP}f&\cellcolor
\ttfamily Vernier&\cellcolor{colT}ø&\cellcolor{colV}e&\cellcolor{colP}f&\cellcolor
\ttfamily Laconnex&\cellcolor{colT}ø&\cellcolor{colV}e&\cellcolor{colP}f&\cellcolor
\ttfamily Veyrier&\cellcolor{colT}ø&\cellcolor{colV}e&\cellcolor{colP}f&\cellcolor
\ttfamily Hermance&\cellcolor{colT}ø&\cellcolor{colV}e&\cellcolor{colP}f&\cellcolor
\ttfamily Semsalves&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}ç&\cellco
\ttfamily Montbovon&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}ç&\cellco
\ttfamily Arconciel&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}ç&\cellco
\ttfamily Avry-sur-Matran&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}ç&
\ttfamily Courtepin&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}ç&\cellco
\ttfamily Dompierre&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}ç&\cellco
\ttfamily Murist&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}ç&\cellco
\ttfamily Sugiez&\cellcolor{colK}f&\cellcolor{colV}æ&\cellcolor{colS}f&\cellcolor
\ttfamily Montalchez&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}f&\cellcolor
\ttfamily Boudry&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}s&\cellcolor
\ttfamily Corcelles&\cellcolor{colK}f&\cellcolor{colV}a&\cellcolor{colS}s&\cellco
\ttfamily Landeron&\cellcolor{colK}f&\cellcolor{colV}a&\cellcolor{colS}s&\cellco
\ttfamily Savagnier&\cellcolor{colK}f&\cellcolor{colV}a&\cellcolor{colS}s&\cellco
\ttfamily Côte-aux-Fées&\cellcolor{colK}t&\cellcolor{colV}a&\cellcolor{colS}f&\cellco
\ttfamily Noiraigue&\cellcolor{colK}f&\cellcolor{colV}a&\cellcolor{colS}f&\cellco
\ttfamily Chaux-du-Milieu&\cellcolor{colK}f&\cellcolor{colV}a&\cellcolor{colS}s&
\ttfamily Cerneux-Péquignot&\cellcolor{colK}f&\cellcolor{colV}e&\cellcolor{colS}s
\ttfamily Lamboing&\cellcolor{colK}f&\cellcolor{colV}a&\cellcolor{colS}s&\cellco
\ttfamily Orvin&\cellcolor{colK}f&\cellcolor{colX}-&\cellcolor{colS}s&\cellcolor
\ttfamily Plagne&\cellcolor{colK}f&\cellcolor{colX}-&\cellcolor{colS}s&\cellcolor
\ttfamily Sombeval&\cellcolor{colK}f&\cellcolor{colX}-&\cellcolor{colS}s&\cellco
\ttfamily Court&\cellcolor{colK}f&\cellcolor{colX}-&\cellcolor{colS}s&\cellcolor
\ttfamily Vermes&\cellcolor{colK}f&\cellcolor{colX}-&\cellcolor{colS}s&\cellcolor
\ttfamily Develier&\cellcolor{colK}f&\cellcolor{colV}ø&\cellcolor{colS}s&\cellco
\ttfamily Cerlatez&\cellcolor{colK}f&\cellcolor{colV}æ&\cellcolor{colS}s&\cellco
\ttfamily Courtedoux&\cellcolor{colK}f&\cellcolor{colX}-&\cellcolor{colS}s&\cell
&\color{white}XXX&\color{white}XXX&\color{white}XXX&\color{white}XXX&\color{white}XXX&\color{white}
\endtabular

}

\endpspicture}

```

10 Linguistischer Hintergrund zum Sprachvergleich

10.1 Phylogenetische Rekonstruktion

10.1.1 Innere und äußere Sprachgeschichte

Georg von der Gabelentz (1840-1893) und die Sprachgeschichte

Der Zweig der Sprachforschung, der uns hier beschäftigt, hat es zunächst mit den trockensten Einzelthatsachen zu thun: Sind die Sprachen A und B miteinander verwandt, und in welchem Grade? Giebt es dieses Wort oder jene Form in der und der Sprache oder in der und der Zeit der Sprachgeschichte? [...] Welche Gesetzmässigkeit herrscht in den lautlichen Abweichungen? Besteht im einzelnen Falle Urgemeinschaft oder Entlehnung? (Gabelentz 1891: 145)

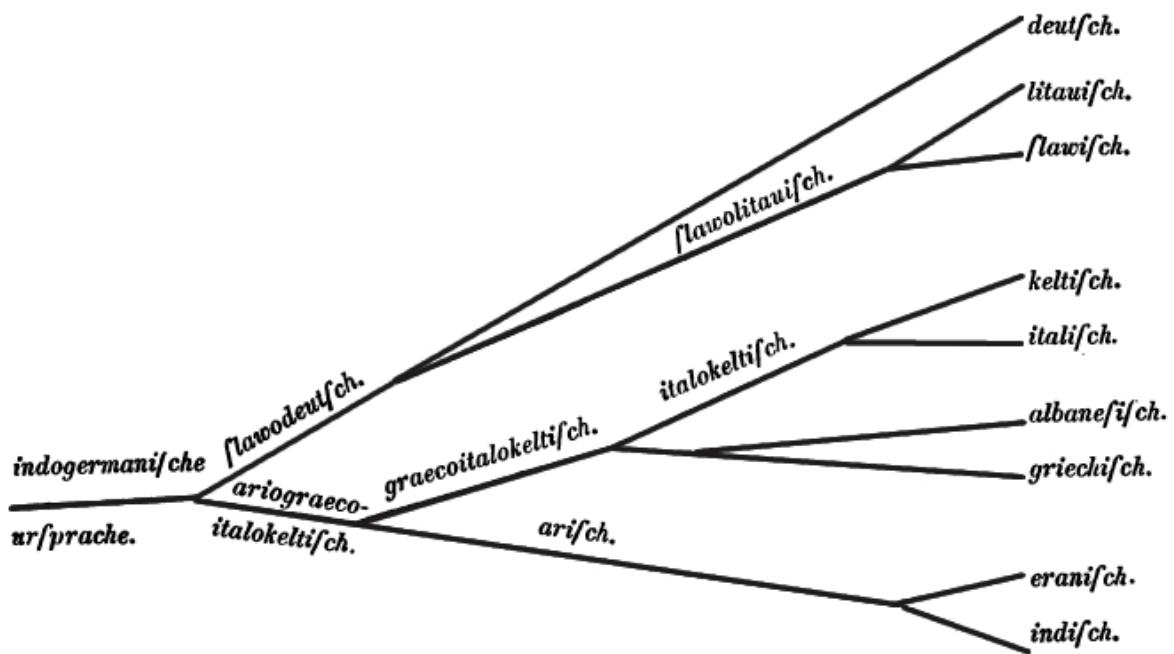
Alles das klingt und ist auch wirklich sehr trocken. Was die menschliche Re- de im Innersten bewegt, was sonst die Wissenschaft von den Sprachen der Völker zu einer der lebensvollsten macht, das tritt hier zunächst zurück: nur einige ihrer Ausläufer ranken in das Seelen- und Sittenleben der Völker hinüber. Der einzelsprachliche Forscher kann gar nicht schnell genug die fremde Sprache in's eigene Ich aufnehmen: der Sprachhistoriker steht draussen vor seinem Gegenstande: hier der Anatom, da der Cadaver. (Gabelentz 1891: 145)

Wir werden, um Missverständnisse zu vermeiden, gut thun, zwischen äusserer und innerer Sprachgeschichte zu unterscheiden. Die äussere Geschichte einer Sprache ist die Geschichte ihrer räumlichen und zeitlichen Verbreitung, ihrer Verzweigungen und etwaigen Mischungen (Genealogie). Die innere Sprachgeschichte erzählt und sucht zu erklären, wie sich die Sprache in Rücksicht auf Stoff und Form allmählich verändert hat. (Gabelentz 1891: 146)

10.1.2 Stammbäume und Wellen

Bäume August Schleicher war eine herausragende Persönlichkeit in der Geschichte der Linguistik. Wir verdanken ihm insbesondere die sogenannte *Stammbaumtheorie*, die er in zwei frühen Werken erstmals im Jahre 1853 veröffentlichte (Schleicher 1853a, Schleicher 1853b). Schleichers Theorie zur äusseren Sprachgeschichte war wahrscheinlich direkt beeinflusst von František Čelakovský (1799 – 1852), den er während einer Professur in Prag kennengelernt hatte, und der noch vor Schleicher einen ersten Stammbaum der slawischen Sprachen veröffentlichte (Čelakovský 1853).

Die ältesten teilungen des indogermanischen bis zum entstehen der grundsprachen der den sprachstamm bildenden sprachfamilien lassen sich durch folgendes schema anschaulich machen. Die länge der linien deutet die zeitdauer an, die entfernung der- selben von einander den verwantschaftsgrad. (Schleicher 1861: 6)

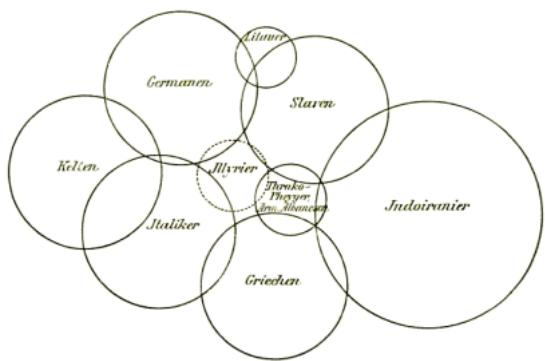


Darstellung aus: Schleicher (1861: 6)

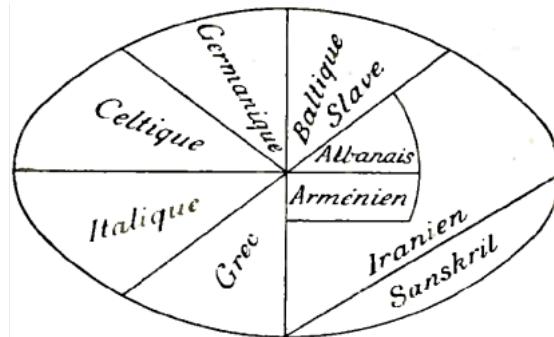
Wellen Nicht lange, nachdem August Schleicher seine berühmte Stammbaumtheorie erstmals postuliert hatte, regte sich Widerspruch in den Kreisen der Indogermanisten und historischen Linguisten. Am bekanntesten ist in diesem Zusammenhang das Werk von Johannes Schmidt (1843 – 1901), der die Stammbaumtheorie verwarf, und an ihrer Stelle seine nicht minder berühmte *Wellentheorie* propagierte.

Ich möchte an seine Stelle das Bild der Welle setzen, welche sich in konzentrischen mit der Entfernung vom Mittelpunkte immer schwächer werdenden Ringen ausbreitet. Dass unser Sprachgebiet keinen Kreis bildet, sondern höchstens einen Kreissektor, dass die ursprünglichste Sprache nicht im Mittelpunkte, sondern an dem einen Ende des Gebietes liegt, tut nichts zur Sache. Mir scheint auch das Bild einer schiefen Ebene vom Sanskrit zum keltischen in ununterbrochener Linie geneigten Ebene nicht unpassend. (Schmidt 1872: 27)

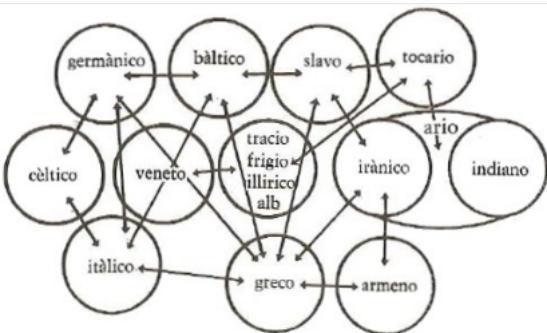
Netze und anderes Gewirr Das größte Problem von Schmidts Wellentheorie war, dass niemand genau wusste, wie er die äußere Sprachgeschichte denn nun schematisch darstellen sollte. Und so finden wir im Laufe der Geschichte eine Vielzahl von Versuchen zur Visualisierung der Wellentheorie.



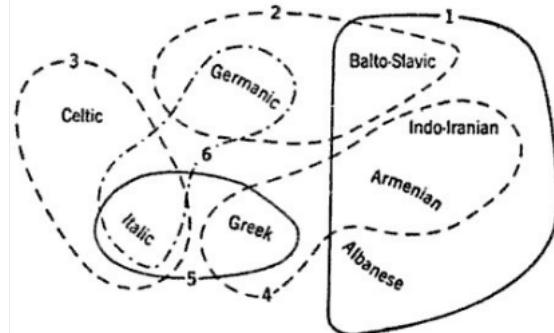
(a) Hirt (1905)



(b) Meillet (1908)



(c) Bonfante (1931)



(d) Bloomfield (1933)

10.1.3 Darstellung und Analyse

Darstellung von Daten und Darstellung von Geschichte Wir müssen unterscheiden zwischen Schemata (seien es Wellen oder Bäume) zur Darstellung von Daten, und Schemata zur Darstellung von Geschichte (vgl. die Unterscheidung zwischen data-display und evolutionary in Morrison 2011: 42f).

Schleichers Baum von 1861 ist dabei ein klares Beispiel für ein Schema, das den Anspruch hat, ein geschichtliches Schema zu sein, während die Beispiele für die Visualisierung der Wellentheorie wohl eher als Datendarstellungsschemata bezeichnet werden sollten, da sie nicht für sich in Anspruch nehmen, äußere Sprachgeschichte zu modellieren. Schemata zur Darstellung von Daten können unter Umständen in Schemata zur Darstellung von Geschichte überführt werden, jedoch hängt die Überführbarkeit davon ab, ob die Daten die Rekonstruktion von Geschichte auch erlauben.

Welche Daten sind es, die auf die Geschichte schließen lassen?

Im ganzen ist also nur wenig, was aus den spezielleren Übereinstimmungen zwischen einzelnen von den acht Hauptgruppen [...] mit grösserer Wahrscheinlichkeit entnommen werden kann. Und jedenfalls treten [...] nirgends speziellere Gemeinsamkeiten, die als gemeinsame Neuerungen erscheinen, in so grosser Anzahl entgegen, dass man auf Grund derselben die betreffenden Sprachzweige in derselben Art zu Einheiten zusammenschliessen dürfte [...]. (Brugmann 1904[1970]: 21f)

10.2 Lexikostatistik

10.2.1 Hintergrund und Grundannahmen

Hintergrund Die Lexikostatistik stellt ein statistisch basiertes Verfahren zur Ermittlung von Verwandtschaftsbeziehungen zwischen Sprachen (und damit zur phylogenetischen Rekonstruktion) dar. Sie wurde von Morris Swadesh (1909 – 1967) in einer Reihe von Artikeln zu Beginn der 50er Jahre des 20. Jahrhunderts vorgestellt und weiterentwickelt (Swadesh 1950, Swadesh 1952, Swadesh 1955). In der Folgezeit mehrte sich jedoch die Kritik an der Methode (Bergsland und Vogt 1962, Hoijer 1956, Rea 1973) und kam am Ende aus der Mode.

Grundsätzlich werden im Rahmen der Lexikostatistik historisch relevante Gemeinsamkeiten zwischen Sprachen ausgezählt. Die zugrunde liegenden Zahlen können dann weiterverwendet werden, um genealogische Bäume automatisch zu rekonstruieren, oder um (unter der Annahme konstanten Wandels) Sprachspaltungszeitpunkte zu ermitteln. Die Methode erlebte zu Beginn des 21. Jahrhunderts eine Wiedergeburt im Rahmen neuer quantitativer biologischer Ansätze, mit deren Hilfe genealogische Bäume automatisch aus spezifischen Sprachdaten gewonnen werden können (Atkinson und Gray 2006, Gray und Atkinson 2003).

Grundannahmen Die Grundannahmen der Lexikostatistik wurden in einer Vielzahl von Arbeiten besprochen (Gudschinsky 1956, Sankoff 1969). Sie lassen sie sich in etwa wie folgt zusammenfassen (vgl. Geisler und List im Erscheinen):

- The lexicon of every human language contains words which are relatively resistant to borrowing and relatively stable over time due to the meaning they express: these words constitute the *basic vocabulary* of languages.
- *Shared retentions* in the basic vocabulary of different languages reflect their degree of *genetic relationship*.

10.2.2 Praktische Umsetzung

Die praktischen Arbeitsschritte

1. **Compilation:** Compile a list of basic vocabulary items (a Swadesh-list).
2. **Translation:** Translate the items into the languages that shall be investigated.
3. **Cognate Judgments:** Search the language entries for cognates.
4. **Coding:** Convert the cognate information into a numerical format.
5. **Computation:** Perform a computational analysis (cluster analysis, tree calculation) of the numerical data, which allows to make conclusions regarding the phylogeny of the languages under investigation.

Diagnostische Testlisten im Concepticon

10.2.3 Kritik

Wichtigste Kritikpunkte in der Literatur

- **Entlehnung:** Unentdeckte Entlehnungen können die Ergebnisse verfälschen.
- **Aussagekraft:** Lexikalische Ersetzung ist als Prozess nicht aussagekräftig für Sprachgeschichte.
- **Fehleranfälligkeit:** Die Methode ist fehleranfällig, da die Daten auf eine problematische Weise erstellt werden.

Entscheidender Kritikpunkt Neuere Vergleiche haben dabei zeigen können, dass die Daten, die von Forscherteams unabhängig produziert werden, derartig große Unterschiede aufweisen, dass dies zu Unterschieden von über 30% in den Daten automatisch berechneten Baumtopologien führt (Geisler und List im Erscheinen). Die größten Probleme liegen dabei weniger im Bereich der Kognatenzuweisungen (Schritt 3, obwohl auch dieser problematisch ist), sondern bereits im Bereich der Übersetzung (Schritt 2).

Bereits hier zeigen sich große Unterschiede zwischen unabhängig erstellten Datensätzen, die zeigen, dass mangelnde Kompetenz der Forscher in den Einzelsprachen, aber auch mangelnde Beschreibung der Konzepte in den Konzeptlisten, zu einer Vielzahl von Unterschieden bereits in den Ausgangsdaten führen können.

10.2.4 Komparanda

Distanzbasierte Ansätze Aus den Daten werden Distanzwerte zwischen den zu vergleichenden Taxa (in unserem Falle Sprachen) abgeleitet. Distanzwerte sind dabei beliebige Zahlen zwischen 0 (Identität der Taxa) und ∞ (Nichtidentität der Taxa), wobei Taxa, die weiter voneinander entfernt sind, einen größeren Distanzwert zugewiesen bekommen, als Taxa, die einander näher liegen.

10.3 Computergestützte Rekonstruktion

Charakterbasierte Ansätze Die Daten werden nicht in Distanzmatrizen transformiert. Stattdessen wird jedes Taxon in Bezug auf eine Reihe von Eigenschaften charakterisiert. Für jede dieser Eigenschaften kann ein Taxon dabei verschiedene Zustände aufweisen. Die einfachste Zustandsbeschreibung ist dabei die Anwesenheit oder Abwesenheit der jeweiligen Eigenschaft (dargestellt in einer Binärmatrix). Komplexere Zustände können den Taxa jedoch ebenfalls zugewiesen werden.

Distanzbasierte versus charakterbasierte Ansätze

	L_1	L_2	L_3
L_1	0	10	20
L_2	10	0	30
L_3	20	30	0

(a) Distanzmatrix

Character	C_1	C2	C_3	C_4	C_5
L_1	0	1	1	1	0
L_2	1	1	0	1	0
L_3	0	0	0	1	1

(b) Merkmalsmatrix

10.3.1 Algorithmen

Allgemeines Vorweg Um aus den in Distanzform oder Charakterform kodierten Daten mit Hilfe des Computers einen Stammbaum abzuleiten, müssen diese geclustert werden. Hierzu sind in der Biologie verschiedene Verfahren entwickelt worden. Es gibt eine Vielzahl verschiedenster Softwareprogramme, welche

diese Aufgabe erfüllen. Den einzelnen Anwendungen innerhalb dieser Software liegen dabei verschiedene Algorithmen zugrunde, welche aus den Daten sukzessive einen Baum erzeugen, oder aus einer Vielzahl möglicher Bäume den wahrscheinlichsten Baum ermitteln.

- **Neighbor-Joining:** Clusterverfahren für Distanzdaten, entwickelt von Saitou and Nei (1987), welches sich insbesondere deshalb großer Beliebtheit erfreut, weil es eine sehr geringe Laufzeit hat. Um einen schnellen Überblick über die Daten zu bekommen, ist es daher sehr gut geeignet.
- **Likelihood-Verfahren:** Verfahren für Charakterdaten, die auf stochastischen Verfahren beruht und entweder alle möglichen Bäume auf ihre Wahrscheinlichkeit vergleichen (“maximum likelihood”, Nunn 2011:33-35), oder ein möglichst großes Sample “guter” Bäume automatisch schätzt (“bayesian methods”, Nunn 2011:35-38).
- **Maximale Parsimonie:** Verfahren, welches den evolutionären Baum errechnet, der am wenigsten evolutionären Wandel benötigt, um die Daten zu erklären (Nunn 2011:30-33).

Software

- LingPy: Bietet distanzbasierte Clusterverfahren, unter anderem Neighbor-Joining (Saitou und Nei 1987) und UPGMA (Sokal und Michener 1958).
- MrBayes: Bietet bayesianische Methoden, die vor allem in der Biologie verwendet werden (Ronquist et al. 2009).
- SplitsTree: Bietet distanzbasierte Clusterverfahren, aber vor allem auch distanzbasierte Netzwerkanalysen, wie zum Beispiel NeighborNet (Bryant und Moulton 2002).

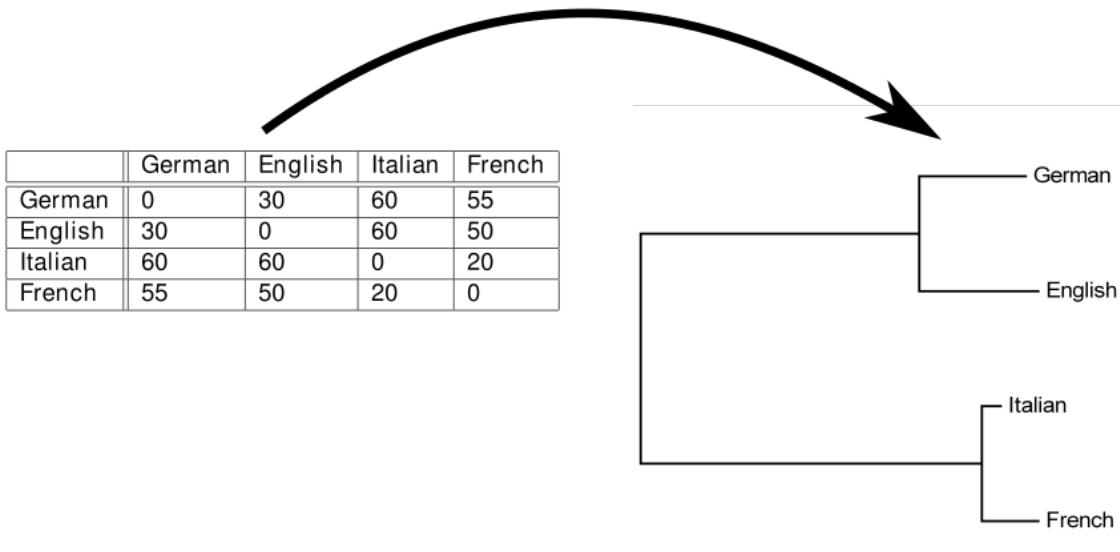
10.3.2 Formate

Allgemeines Vorweg Als Eingabedaten werden in den phylogenetischen Analysen im Rahmen der historischen Linguistik zumeist Swadesh-Listen (vgl. Swadesh 1950, 1952, 1955) verwendet. Jedes Item (“Bedeutungsslot”) wird dabei zunächst als Charakter angesehen. Diesem wird für jede Sprache ein Zustand zugewiesen, wobei der Zustand als identisch kodiert wird, wenn die jeweiligen Spracheinträge kognat sind.

Distanzberechnung aus Swadeshlisten

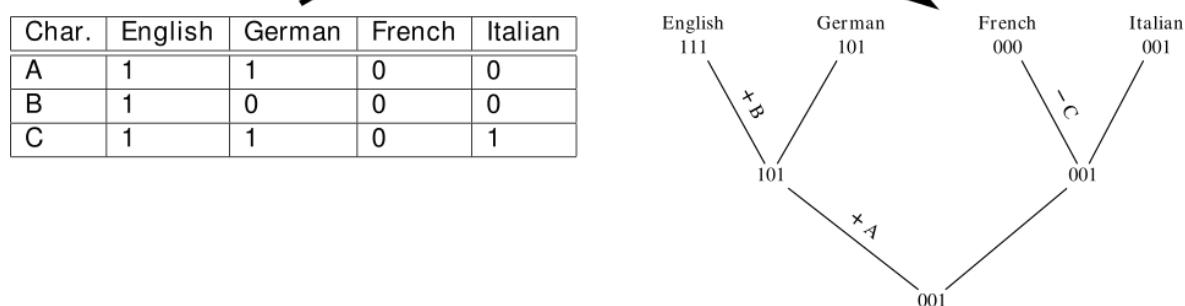
Basic Concept	German	ID	English	ID	Italian	ID	French	ID
HAND	Hand	1	hand	1	mano	2	main	2
BLOOD	Blut	3	blood	3	sangue	4	sang	4
HEAD	Kopf	5	head	6	testa	7	tête	7
TOOTH	Zahn	8	tooth	8	dente	8	dent	8
TO SLEEP	schlafen	9	sleep	9	dormir	10	dormir	10
TO SAY	sagen	11	say	11	dire	12	dire	12
...

Distanzberechnung aus Swadeshlisten



Charaktermodellierung von Kognatensätzen

ID	Proto-Form	Basic Concept	German	English	Italian	French
1	PGM *xanda-	HAND	1	1	0	0
2	LAT <i>mānus</i>	HAND	0	0	1	1
3	PGM *bloda-	BLOOD	1	1	0	0
4	LAT <i>sanguis</i>	BLOOD	0	0	1	1
5	PGM *kappa	HEAD	1	0	0	0
6	PGM *xawbda-	HEAD	0	1	0	0
7	LAT <i>tēsta</i>	HEAD	0	0	1	1
8	PIE *h₂dōnt-	TOOTH	1	1	1	1
9	PGM *slépan-	TO SLEEP	1	1	0	0
10	LAT <i>dormire</i>	TO SLEEP	0	0	1	1
11	PGM *sagjan-	TO SAY	1	1	0	0
12	LAT <i>dicere</i>	TO SAY	0	0	1	1
...



Phylip-Format

```

12
Germa 0.0 0.19 0.11 0.27 0.19 0.26 0.16 0.16 0.19 0.22 0.11 0.08
Engli 0.19 0.0 0.14 0.27 0.19 0.22 0.16 0.16 0.26 0.22 0.11 0.14
Dutch 0.11 0.14 0.0 0.29 0.21 0.27 0.14 0.18 0.24 0.24 0.13 0.09
Icela 0.27 0.27 0.29 0.0 0.08 0.21 0.14 0.11 0.27 0.08 0.22 0.22
Nynor 0.19 0.19 0.21 0.08 0.0 0.13 0.06 0.03 0.22 0.06 0.14 0.14
Riksm 0.26 0.22 0.27 0.21 0.13 0.0 0.13 0.09 0.29 0.16 0.21 0.21

```

	Swedi	0.16	0.16	0.14	0.14	0.06	0.13	0.0	0.03	0.19	0.09	0.11	0.11
Danis	0.16	0.16	0.18	0.11	0.03	0.09	0.03	0.0	0.19	0.06	0.11	0.11	
Gothi	0.19	0.26	0.24	0.27	0.22	0.29	0.19	0.19	0.0	0.22	0.18	0.18	
OIcel	0.22	0.22	0.24	0.08	0.06	0.16	0.09	0.06	0.22	0.0	0.14	0.18	
OEngl	0.11	0.11	0.13	0.22	0.14	0.21	0.11	0.11	0.18	0.14	0.0	0.06	
OHGer	0.08	0.14	0.09	0.22	0.14	0.21	0.11	0.11	0.18	0.18	0.06	0.0	

Nexus-Format

```
#nexus
BEGIN Taxa;
DIMENSIONS ntax=12;
TAXLABELS 'Germa' 'Engli' 'Dutch' 'Icela' 'Nynor' 'Riksm' 'Swedi' 'Danis'
'Gothi' 'OIcel' 'OEngl' 'OHGer';
END; [Taxa]
BEGIN Characters;
DIMENSIONS nchar=61;
FORMAT
datatype=STANDARD
missing=?
gap=-
symbols="01"
labels=left
transpose=no
interleave=no
;
MATRIX
'Germa' 01111111111111111111111111111111111111111111111111111111111111110000000000000000000000000000000000
'Engli' 0100011111111111111110111111111010110111111111111100000000000000000000000000000000000000
'Dutch' 010111111111111111111111111111111010111111111111111100101100000000000000000000000000
'Icela' 01001111101111111101111110111101111011111111110001001111111110000
AH
'Nynor' 01001111101111111101111111110111110111111111100010011110010000000
'Riksm' 01000101101110101111110111110111111110001001010010010000
'Swedi' 0100111110111111111111111111111111110111110111111111110001110101000000000000
'Danis' 01001111101111111111111111111111111101111101111111111100010010100100000000
'Gothi' 010011101111111111111111111101111011111111111000010010000000001111
'OIcel' 010011111011111111111111111111111111011110111111111110001001111010101000
'OEngl' 01011111111111111111111111111111111101111011111111101010000000000001000
'OHGer' 010111111111111111111111111111111111011111111111111111010000100000000000000
;
```

11 Automatischer Sprachvergleich mit Python

11.1 Automatischer Sprachvergleich

11.1.1 Sequenzdistanzen

Von Alinierungen zu Distanzwerten Wenn man Sequenzen (Wörter im Falle der Linguistik) aliniert hat, dann kann man auch ihre Distanz zueinander berechnen.

nen. Die ist implizit ja bereits ein Teil der Berechnung der Alinierung, und sie kann entsprechend auch einfach weiterverwendet werden, wobei man vorsichtig sein sollte, was die Aussage einer solchen Distanz betrifft.

Die normalisierte Levenshtein-Distanz Eine der bekanntesten und am häufigsten verwendeten Distanzen zwischen Strings ist die *Levenshtein-Distanz*, die grundlegend die Anzahl der *Editier-Schritte* beschreibt, die notwendig sind, um eine Sequenz in die andere zu überführen. Um eine bessere Vergleichbarkeit zu gewährleisten, wird häufig neben der normalen *Levenshtein-Distanz*, die ein ganzzahliger Wert ist, auch die “normalisierte Levenshtein-Distanz” berechnet, bei der man die “normale” Distanz durch die Länge des größeren Strings teilt.

```
>>> from lingpy import *
>>> seqA = 'levensthein'
>>> seqB = 'levenschtein'
>>> d = edit_dist(seqA,seqB)
>>> d
1
>>> def norm_ed(a,b):
...     return edit_dist(a,b) / max(len(a), len(b))
...
>>> norm_ed(seqA, seqB)
0.0833333333333333
>>> edit_dist(seqA, seqB, normalized=True)
0.0833333333333333
```

11.1.2 Kognatenerkennung

Automatisches Erkennen von kognaten Wörtern Kognaten sind, daran sei noch mal erinnert, Wörter, die auf einen gemeinsamen Vorgänger zurückgehen (wie bspw. Deutsch *Hand* und English *hand*). Automatisch können wir eine recht einfache Heuristik entwickeln, um festzustellen, ob zwei Wörter kognat sind (was nicht heißt, dass sie das auch wirklich sind!). Wir können einfach sagen, dass, wenn immer zwei Wörter sich mehr ähneln als gewöhnlich, wir annehmen, dass diese kognat sind. Die Ähnlichkeit können wir dabei natürlich unterschiedlich definieren!

Die Lautklassenheuristik Eine erste einfache Heuristik besagt, dass zwei Wörter immer dann als kognat klassifiziert werden, wenn sie in den ersten beiden Lautklassen, die Konsonanten sind, übereinstimmen (Turchin et al. 2010).

```
>>> seqA = sampa2uni('Tri')
>>> seqB = sampa2uni('drai')
>>> seqA, seqB
('θri', 'drai')
>>> clsA = tokens2class(ipa2tokens(seqA), 'dolgo')
>>> clsB = tokens2class(ipa2tokens(seqB), 'dolgo')
>>> clsA = ''.join(clsA).replace('V', '')
>>> clsB = ''.join(clsB).replace('V', '')
>>> clsA[:2] == clsB[:2]
```

```

True
>>> clsA, clsB
('TR', 'TR')
>>> turchin(seqA, seqB)
0
>>> turchin(seqA, 'test')
1

```

Schwellenwerte Anstelle des Kriteriums von Turchin et al. (2010) können wir natürlich auch andere Verfahren verwenden. Zum Beispiel können wir sagen, dass ab einem bestimmten Schwellenwert der normalisierten Levenshtein-Distanz zwei Wörter nicht mehr kognat sind:

```

>>> def cognate(seqA, seqB, threshold=0.4):
...     if edit_dist(seqA, seqB, normalized=True) > threshold:
...         return 1
...     return 0
...
>>> seqA = sampa2uni('t_h0xt@r')
>>> seqB =
KeyboardInterrupt
>>> seqA = sampa2uni('t_h0xt_h@r')
>>> seqB = sampa2uni('d0:t_h@r')
>>> cognate(seqA, seqB)

```

11.1.3 Phylogenetische Rekonstruktion

Von Distanzen zu Bäumen Wenn wir ermittelt haben, welche Wörter in welchen Sprachen miteinander kognat sind, können wir Distanzen zwischen ganzen Sprachen berechnen. Dazu zählen wir einfach alle kognaten Wörter in unserem Sample und teilen dann die Anzahl der kognaten Wörter durch die Gesamt-Anzahl der Wörter und ziehen diesen Wert von 1 ab (ansonsten hätten wir eine Ähnlichkeit und keine Distanzen).

Kleines Beispiel zur Distanzberechnung

```

from lingpy import *
# die daten
data = dict(
    german = ['hant', 'fu:s', 'kɔpf'],
    english = ['hænd', 'fʊt', 'hɛd'],
    dutch = ['hant', 'vut', 'kɔp']
)
# die sprachen
taxa = ['german', 'english', 'dutch']
# die distanzmatrix
matrix = [[0 for i in range(3)] for j in range(3)]
for i,k in enumerate(taxa):
    for j,l in enumerate(taxa):
        # wir müssen nur ein mal pro sprachpaar vergleichen

```

```

        if i < j:
            score = 0
            for seqA, seqB in zip(data[k], data[l]):
                score += edit_dist(seqA, seqB, normalized=True)
            score = score / 3
            matrix[i][j] = score
            matrix[j][i] = score
    # der baum
    tree = upgma(matrix, taxa)
    # ascii-art mit Hilfe von LingPy
    print(Tree(tree).asciiArt())

```

Kleines Beispiel zur Distanzberechnung

```

$ python distances.py
      /-english
-root---|
          |      /-german
          \edge.0--|      \-dutch

```

11.2 Sprachvergleich mit LingPy

11.2.1 Eingabeformate

Basisformat für Wortlisten

ID	CONCEPT	COUNTERPART	IPA	DOCULECT	COGID
1	hand	Hand	hant	German	1
2	hand	hand	hænd	English	1
3	hand	рука	ruka	Russian	2
4	hand	рука	ruka	Ukrainian	2
5	leg	Bein	bain	German	3
6	leg	leg	lɛg	English	4
7	leg	нога	noga	Russian	5
8	leg	нога	noha	Ukrainian	5
9	Woldemort	Waldemar	valdemar	German	6
10	Woldemort	Woldemort	woldemɔrt	English	6
11	Woldemort	Владимир	vladimir	Russian	6
12	Woldemort	Володимир	volodimir	Ukrainian	6
13	Harry	Harald	haralt	German	7
14	Harry	Harry	hæri	English	7
15	Harry	Гарри	gari	Russian	7
16	Harry	Гарпи	hari	Ukrainian	7

Key-Value-Erweiterung des Basisformats

#	ID	CONCEPT	COUNTERPART	IPA	DOCULECT	COGID	ALIGNMENT
	1	hand	Hand	hant	German	1	
	2	hand	hand	hænd	English	1	

3	hand	рука	рука	Russian	2	
...

Darstellung von Alinierungen

#	ID	CONCEPT	COUNTERPART	IPA	DOCULECT	COGID	ALIGNMENT
...
9	Woldemort	Waldemar	valdemar	German	6	v a l - d e m a	
10	Woldemort	Woldemort	woldemort	English	6	w o l - d e m o	
11	Woldemort	Владимир	vladimir	Russian	6	v - l a d i m i	
12	Woldemort	Володимир	volodimir	Ukrainian	6	v o l o d i m i	
...

Kognatenerkennung mit LexStat Kognatenerkennung in LingPy lässt sich mit Hilfe des LexStat-Moduls durchführen. Basierend auf einer Datei, die das oben beschriebene EingabefORMAT aufweist und zwingend die Spalten “CONCEPT”, “IPA”, und “DOCULECT” enthalten muss, kann man mit Hilfe des LexStat-Moduls Kognaten auf verschiedene Art und Weise berechnen, diese Werte dann in Distanzen umwandeln, und aus den Distanzen auch direkt einen Baum berechnen.

Beispiel zur Kognatenerkennung mit LexStat

```
>>> from lingpy import *
>>> lex = LexStat('data/harry.tsv')
>>> lex.cluster(method='turchin')
>>> lex.cluster(method='turchin')
|+++++-----+-----+-----+-----+-----+-----+-----+-----+-----+
>>> lex.calculate(tree, ref="turchinid")
>>> print(lex.tree.asciiArt())
                  /-English
                  /edge.0--|
                  |          \-German
-root---|          |          /-Russian
          |          \edge.1--|
          |                      \-Ukrainian
```

Alinierung von Kognatensets Es ist sinnvoll, die automatisch ermittelten Kognaten auch zu alinieren, da man sich so am besten vergewissern kann, dass man auch keine falschen Kognaten ermittelt oder richtige Kognaten übersehen hat. Hierfür bietet sich das SCA-Modul von LingPy an, welches als Eingabe eine Wortliste nimmt und neben den für LexStat geforderten Spalten “DOCULECT”, “CONCEPT” und “IPA” eine weitere Spalte erfordert, die dem Programm mitteilt, wo die Kognaten-Identifikationsnummern sind. Normalerweise werden diese Spalten in LingPy nach der Methode beziffert, die ihnen zugrunde liegt (“turchin” == “turchinid”, “sca” == “scaid”, etc.), wobei der Name “cogid” gewöhnlich für eine von Experten ermittelte Kognatenzuweisung reserviert wird.

Beispiel für die Alinierung von Kognatensets

```
>>> from lingpy import *
>>> alm = Alignments('data/harry.tsv', ref="cogid")
>>> alm.align()
| ----- ALIGNMENTS ----- |
>>> alm.output('html', filename='data/harry')
```

11.2.2 Analysen

Ausgabe der Daten in HTML-Format

11.2.3 Ausgabeformate

Grundlegendes zu Ausgabeformaten in LingPy Daten in LingPy können in verschiedenste Formate exportiert werden. Grundlegend unterscheiden können wir dabei zwei Formattypen:

- Endformate, die entweder für Publikationen oder zur manuellen Untersuchung von Ergebnissen verwendet werden können (Plots, Grafiken), und
- Übergangsformate, die zur Weiterverwendung in alternativen Softwarepackungen verwendet werden können.

Spezifizieren von Ausgabeformaten in LingPy

```
# grundlegenes Format des Kommandos
wordlist.output(DTYPE, filename=NAME)
# Beispiele
## Exportiere nach Phylip (Distanzformat)
wordlist.output('dst', filename="harry")
## Exportiere nach Nexus (Charakterformat)
wordlist.output('paps.nex', filename="harry")
## Exportiere nach HTML (individuelles LingPy-Format)
wordlist.output('html', filename="harry")
## Exportiere den Baum in Newick-Format
wordlist.output('nwk', filename="harry")
## Exportiere zum LingPy-Wordlist-Format
wordlist.output('tsv', filename='harry')
```

Beispiele für die Ausgabeformate Newick-Format:

((English,German),Russian),Ukrainian);

Phylip-Format:

```
4
English    0.00 0.25 0.50 0.50
German     0.25 0.00 0.50 0.50
Russian    0.50 0.50 0.00 0.00
Ukrainian  0.50 0.50 0.00 0.00
```

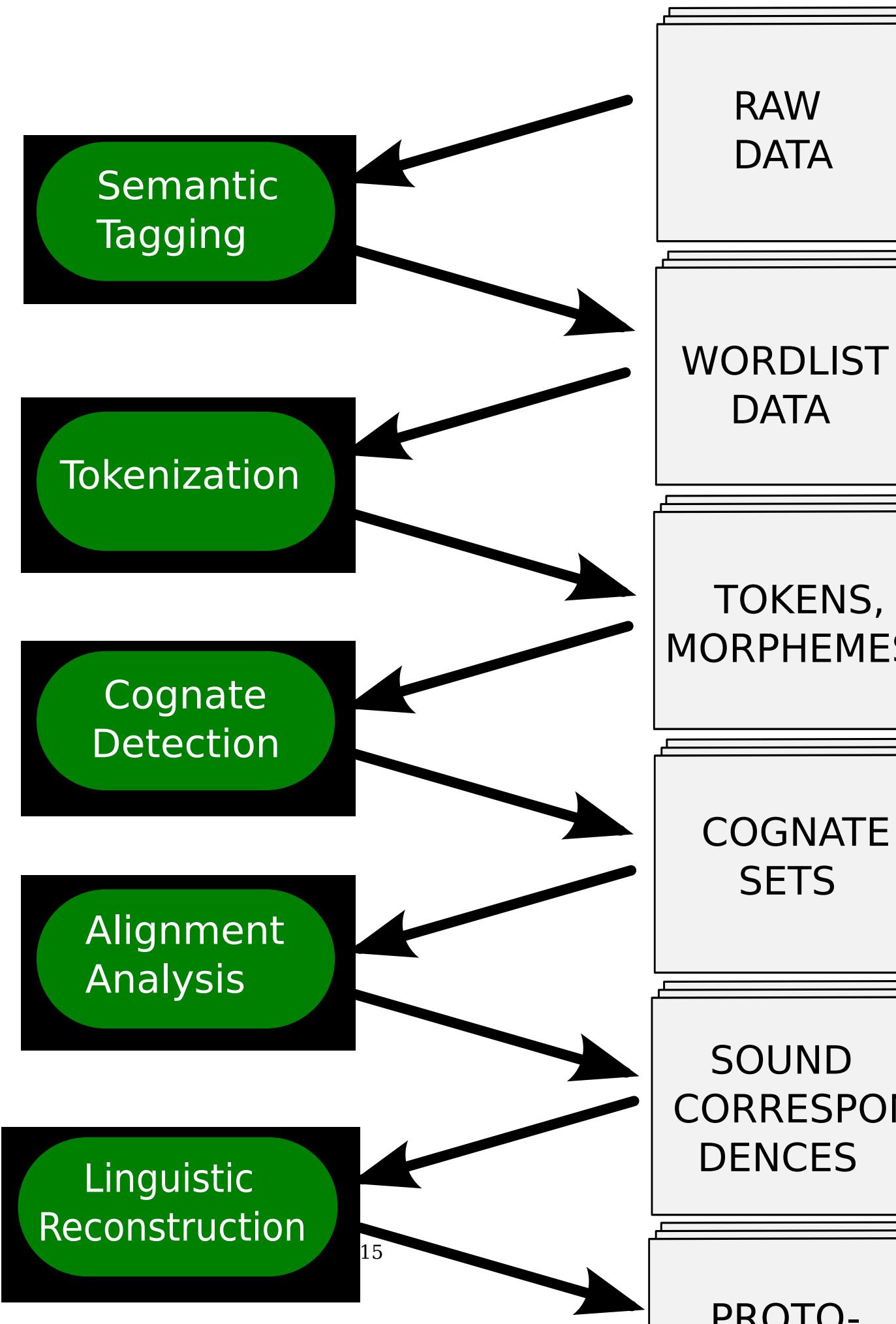
Nexus-Format:

```
#NEXUS
BEGIN DATA;
DIMENSIONS ntax=4 NCHAR=7;
FORMAT DATATYPE=STANDARD GAP=- MISSING=0 interleave=yes;
MATRIX
English    1001011
German     1010011
Russian    0100111
Ukrainian 0100111
;
END;
```

11.3 Workflows

11.3.1 Allgemeines vorweg

Workflows in LingPy Mit Hilfe der Funktionen, die LingPy bietet, lassens ich komplette Workflows zum automatisierten Vergleich von Sprachen entwickeln. Das ist nicht immer einfach, da nur ein Bruchteil der Möglichkeiten von LingPy auch ordentlich dokumentiert ist. Wir wollen aber trotzdem versuchen, ein allgemeines Template zu erstellen, so dass es möglich ist, dieses an individuelle Daten anzupassen und weiterzuverwenden.



Workflows zum Sprachvergleich

11.3.2 Kognatenerkennung mit LingPy

Daten Wir nehmen einen Datensatz zu den chinesischen Dialekten, der als TSV-Datei im LingPy Format vorliegt (File "chinese.tsv").

Die Datei enthält neben den tabularen Daten auch eine JSON-Spezifikation (eine Formaterweiterung in LingPy, die es erlaubt, JSON-Daten mit einzubinden). Wir ignorieren diese Daten jedoch in diesem Zusammenhang.

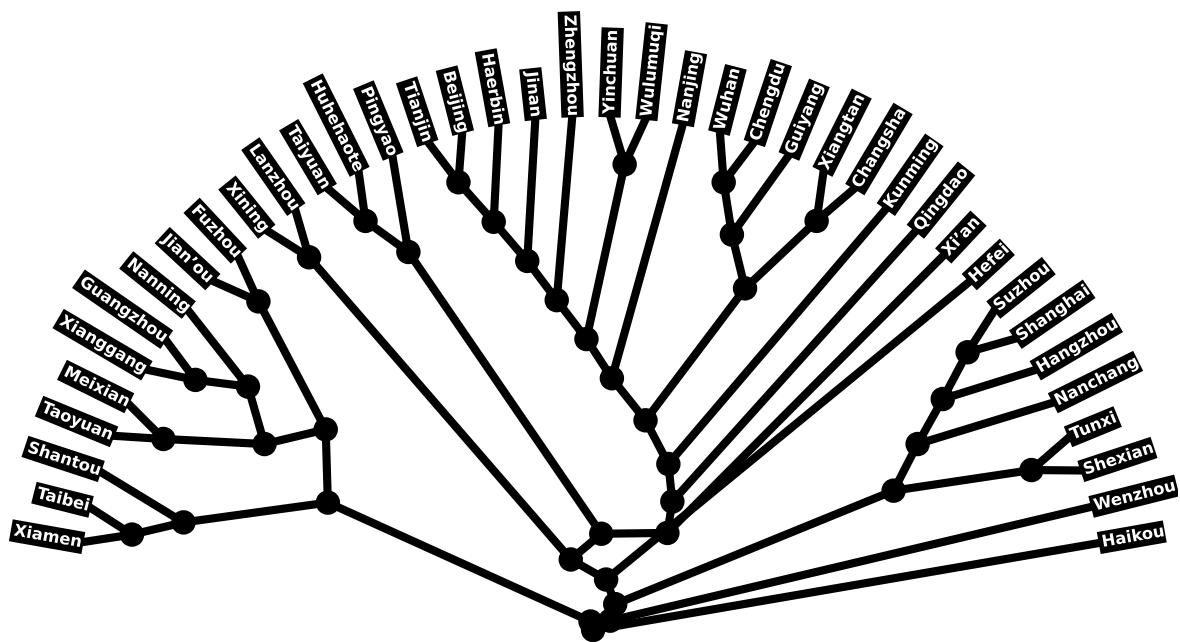
Der Workflow Der Workflow gliedert sich in drei Schritte:

- Kognatenberechnung:
 - Einlesen der Datei in LingPy (LexStat Modul)
 - Berechnen von Kognatensets mit Hilfe von LexStat
 - Auslesen der Datei mit den neu berechneten Daten in TSV-Format
- Berechnen und Plotten eines phylogenetischen Baums
- Berechnung der Alinierungen
 - Einlesen der Datei in LingPy (Alignments Modul)
 - Berechnen der Alinierungen mit Hilfe von Alignments
 - Auslesen der Datei in HTML-Format

Der Code

```
from lingpy import *
# Schritt 1
## 1.1 Einlesen der Daten
lex = LexStat('data/chinese.tsv')
## 1.2 Kognatenerkennung
lex.cluster(method='sca', threshold=0.4)
## 1.3 Auslesen der Daten
lex.output('tsv', filename='data/chinese_lexstat', ignore='all')
# Schritt 2
## 1.1 Berechnen des Baums
lex.calculate('tree', ref="scaid") # scaid sind die automatischen kognaten
## 1.2 Plotten des Baums
from lingpy.convert.plot import plot_tree
plot_tree(lex.tree, degree=160, filename="data/chinese_tree", fileformat="svg")
## 1.3 Schreiben der Distanz-Daten
lex.output('dst', filename="data/chinese_distances")
# Schritt 3
## 1.1 Einlesen der Daten
alm = Alignments('data/chinese_lexstat.tsv', ref="scaid")
## 1.2 Alinierung
alm.align()
## 1.3 Auslesen der Daten in HTML
alm.output('html', filename='data/chinese_alignments')
```

Die Ergebnisse: Der Baum



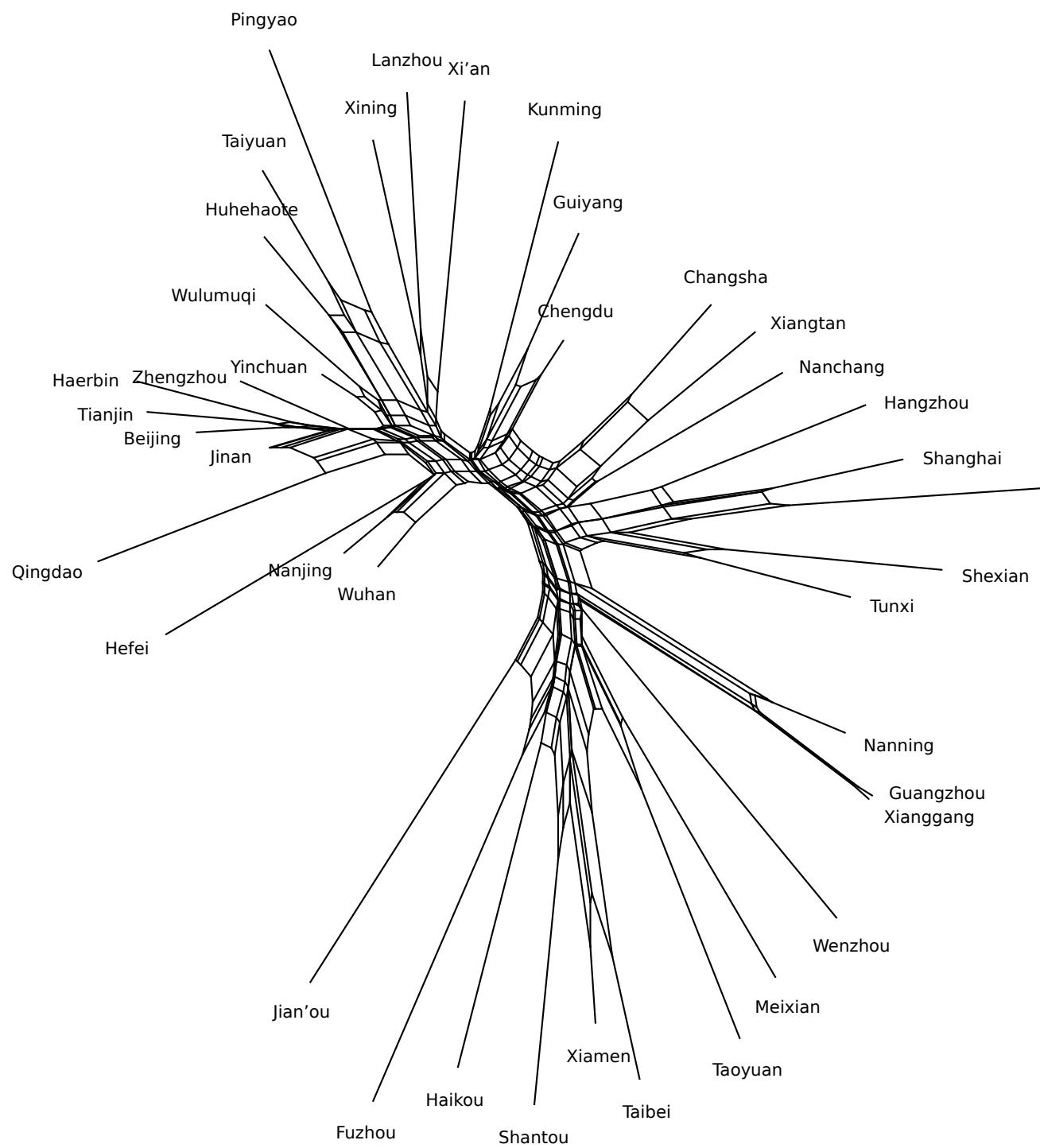
Die Ergebnisse: Die Alinierungen

11.3.3 Integration mit externen Tools

Einlesen der Daten in Splitstree Mit SplitsTree können wir Netzwerke aus Distanzmatrizen berechnen. Da wir die Distanzmatrix ja bereits exportiert haben, genügt es, wenn SplitStree erfolgreich installiert wurde, diese entweder direkt als Textdatei einzulesen, oder den Inhalt der Datei `chinese_distances.dst` zu kopieren und in den Editor von SplitsTree zu pasten.

Das Neighbor-Net der chinesischen Daten

— 0.1 —



12 Geoplots mit JavaScript und D3

12.1 Darstellung geographischer Daten

12.1.1 GeoJson und TopoJson

Grundlegendes GeoJson und TopoJson sind Formate zur einfachen Beschreibung geographischer Daten. Sie bauen beide auf dem JSON-Format auf, welches wir im Laufe des Woche ja schon zuweilen verwendet haben. Beide Formate stellen eine ideale Grundlage für geographische Darstellungen mit JavaScript dar und sind auch ineinander überführbar. Wir werden TopoJson für unser Beispiel verwenden.

GitHub-Integration

- Das tolle an Geo- und TopoJson ist, dass die Formate direkt in GitHub als Karten dargestellt werden.
- Wenn wir uns Beispielsweise die Datei zh-mainland-provinces-topo.json direkt auf GitHub anschauen, dann sehen wir direkt eine Karte, und auch, was auf der Karte zu sehen ist!

Getting Started Grundlegendes zu den Formaten, aber auch zum Gestalten einer Karte kann man im Internet in vielen Beispielen finden. Ich selbst habe meine ersten Erkenntnisse über D3 und die Verwendung von geographischen Daten in JS durch das Beispiel Let's Make a Map von Mike Bostock gesammelt.

Dort beschreibt Bostock auch, wo man gute geographischen Daten finden kann. Diese liegen meist in Form von "shapefiles" vor, die nicht in JS verwendet werden können. Aber es gibt mit mapshaper ein sehr gutes Tool, um die Daten direkt in die JSON-Formate umzuwandeln.

12.1.2 D3

D3 ist eine wunderbare Bibliothek, um Daten zu Visualisieren und interaktive Applikationen zu schreiben. Leider ist sie auch sehr kompliziert und für Anfänger nur schwer zugänglich, da sich hinter dem Kode viele sehr sinnvolle aber auch eigenwillige Konzepte verbergen.

Wir haben in diesem Seminar leider keine Zeit, voll auf D3 und die Möglichkeiten einzugehen. Wir werden uns daher auf ein Beispiel beschränken, in dem es weniger um den Kode als um die Idee der Visualisierung geht.

Denjenigen, die mehr über D3 erfahren möchte, empfehle ich, mit der offiziellen Homepage anzufangen, und sich dann langsam "hochzuarbeiten".

12.2 Eine Beispielapplikation

12.2.1 Idee

Die Daten Wir haben uns schon mit der Kollektion chinesischer Daten befasst und sie sogar bereits mit Hilfe von LingPy aliniert. Nun wollen wir einen Schritt weiter gehen, und die Diversität der Sprachen nicht nur über die einzelnen Alinierungen, sondern auch im Raum darzustellen.

12.2.2 Idee

Die Idee Die Idee ist einfach: Wir plotten alle Dialektpunkte auf eine Karte, und erlauben dann den Benutzern, Konzeptweise Daten aufzurufen und bei Klick auf einen Dialektpunkt sowohl die Verteilung des Kognatensets als auch die Wörter in alinierter Form zu betrachten.

Die Tools

- Python: zur Alinierung der Daten und zur Aufbereitung der Daten in JavaScript-kompatible Formate (JSON, JS Objekte).
- JavaScript: zum Plotten der Daten mit Hilfe von D3.

12.2.3 Vorbereitung

Programmarbeit

- Erstellen der Daten und Überführung in LingPy-Wortlisten-Format
- Durchführung der Alinierungsanalyse mit Hilfe von LingPy
- Exportieren der Daten in JS-Formate (hauptsächlich JS-Objects, die ja wie ein Hash verwendet werden können und identisch mit JSON-Objekten sind).

Hintergrundarbeit

- Geographische Files konnten übernommen werden aus einem GitHub-Repository.
- Der Kode musste nur geringfügig angepasst werden, und die Dialektpunkte ergänzt werden.

Planung der Applikation

- Grundlegende Idee: Teile die Applikation in zwei Teile, einen mit der Karte, einen mit den Alinierungen
- Zur Darstellung der Alinierungen liegt bereits eine JS-Bibliothek vor, die es ermöglicht, Wörter, die segmentiert vorliegen, koloriert darzustellen.
- Erweiterte Ideen wurden in einem nicht ratsamen “trial-and-error-Vefahren” entwickelt (planen ist immer besser als ausprobieren, aber man plant dann am Ende doch immer viel zu wenig...).

12.2.4 Applikation