doc

January 9, 2021

1 Projekt zu 23-TXT-BaCL3

1.1 Bibtex-Referenzen auflösen

Aufgabenstellung:

"Für die Textauszeichnungs- und -gestaltungssprache LaTeX gibt es das BibTeX-Format zur Strukturierung von Literaturangaben. In dieser Aufgabe sollen Sie eine Funktion schreiben, die einen Text entgegennimmt, in welchem Literaturverweise durch (Nachname Jahr) ersetzt werden sollen, sowie an welche ein Abschnitt angefügt werden soll, der die Literaturangabe vollständig aufführt."

1.1.1 Einleitung

Das vorliegende Projekt ist objektorientiert programmiert. Es beginnt mit einem einzigen Objekt, der Klasse *BibTeX*. Dieses bildet die Datenkapsel in der weitere Instanzen (Objekte) enthalten sind. Instanzen bezeichnen hier Attribute (Daten) oder Methoden (Funktionen). Ein Attribut kann ein Instanz- oder Klassen-Attribut sein und über die Form *self.attribute* bzw. *Class.attribute* abgerufen werden. Analog dazu kann eine Methode eine Instanz- oder Klassen-Methode sein, die über die Form *self.method()* respektive *Class.method()* abgerufen wird. Darüber hinaus sind auch statische Methoden möglich, die unabhängig von einer Initialisierung der Klasse funktionieren.

Im Falle dieses Projekts wird die Klasse *BibTeX* mittels "class BibTeX(object):" definiert. Sie umfasst sieben Instanz-Methoden, die wiederum durch das Schema "def method(self):" definiert werden. In dieser Schreibweise repräsentiert *self* , welches der Methode als obligatorisches Argument übergeben wird, die Instanz der Klasse. Die erste dieser Methoden geht auf die Grundform *___init___(self)* zurück und ist eine sogenannte magische Methode. Sie bildet einen Sonderfall, da sie den Konstruktor der Klasse darstellt. Sie umfasst hier die drei Attribute *txt*, *bib* und *output*, die mittels *self* einer Instanz der Klasse zugeordnet werden. Die sechs weiteren Methoden (*check_all_keys*, *check_all_types*, *bib_to_dict*, *check_required_fields*, *transfer* und *bibliography*) bündeln jeweils spezifische Funktionen des Programms, die auf eine Instanz der Klasse angewendet werden können. Sie stehen teilweise in Abhängigkeit von einander, da sie in vier Fällen Produkte anderer Methoden entgegennehmen (→ Instanz-Attribute), und bilden zusammengefasst den Kern des Programms.

1.1.2 Vorgehensweise

Meine anfangs **geplante Vorgehensweise** geht auf eine schrittweise Abstraktion der Aufgabe zurück, bei der ich versucht habe, das Problem mit einem natürlichsprachlichen Algorithmus zu lösen. Einen solchen Algorithmus habe ich zur Orientierung vor Beginn des Programmierens wie folgt niedergeschrieben:

1. Nimm .txt-Datei und .bib-Datei entgegen.
2. Erstelle leere Ausgabedatei
3. Überprüfe .txt-Datei auf Vorkommen des Zitationsbefehls.
    1. Falls Zitationsbefehl gefunden:
        1. Suche Referenz im Zitationsbefehl (*Key*) in der .bib-Datei.
        2. Überprüfe den Literaturtyp und einhergehende Felder
           *(Die in BibTeX verwendeten Literaturtypen (*Entry Types*) gehen mit folgenden Feldern (*Fields*) einher: Referenzart, erforderliche Felder, optionale Felder. Eine definierte Referenzart hat jeweils spezifische erforderliche Felder, die angegeben werden müssen, sowie optionale Felder, die angegeben werden können. Angaben, die nicht der Definition entsprechen, werden ignoriert)*
            1. Ist die Referenzart korrekt? (Falls nicht: Fehlermeldung)
            2. Sind alle erforderlichen Felder der spezifischen Referenzart angegeben? (Falls nicht: Fehlermeldung)
            3. Sind optionale Felder angegeben?
            4. Sind andere Felder angegeben? (Falls ja: ignorieren)
        3. Merke dir, wenn ein *Key* benutzt wurde.
        4. Schreibe in die Ausgabedatei den Inhalt der Textdatei bis zum Zitationsbefehl und ersetzte den Zitationsbefehl durch Zitationsangabe.
        5. Überprüfe .txt-Datei erneut auf Vorkommen des Zitationsbefehls und gehe analog vor. Wenn keiner mehr vorhanden UND gesamter Text der Textdatei in der Ausgabedatei, gehe weiter.
        6. Suche für jeden verwendeten *Key* die einhergehenden Angaben aus der .bib-Datei und hänge an die Ausgabedatei ein Literaturtypen-sensitives Literaturverzeichnis an.
    2. Falls kein Zitationsbefehl gefunden:
        1. Programm beendet

Die **schlussendliche Vorgehensweise** folgt diesem detaillierterem Algorithmus:

1. Erstelle eine Klasse, die das gesamte Programm beinhalten wird.

Initialisierung der Klasse:

___init___

1. **Nimm** einen Pfad zu einer .txt-Datei **entgegen** und lies diese ein.
2. **Nimm** einen Pfad zu einer .bib-Datei **entgegen** und lies diese ein.
3. Frag, ob die Ausgabedatei im Arbeitsverzeichnis erzeugt werden soll. Wenn nicht, **nimm** einen alternativen Pfad **entgegen**. Frag nach dem gewünschten Namen der Ausgabedatei und erstelle sie im gewünschten Pfad.

Funktionen der Klasse:

check_all_keys

1. Finde alle BibTeX-Schlüssel in den Zitationsbefehlen der .txt-Datei.
2. Finde alle BibTeX-Schlüssel in der Literaturdatenbank der .bib-Datei.

3. Prüfe, ob jeder BibTeX-Schlüssel in der .bib-Datei auch in der .txt-Datei vorkommt und erstelle eine Liste der nicht-vorkommenden BibTeX-Schlüssel.
4. Prüfe, ob jeder BibTeX-Schlüssel in der .txt-Datei auch in der .bib-Datei vorkommt.
5. Falls ein BibTeX-Schlüssel mehrfach vorkommt, schließe mit einer Fehlermeldung.
6. Falls ein BibTeX-Schlüssel nicht vorkommt, schließe mit einer Fehlermeldung.
7. **Gib** die Liste der nicht in der .txt-, aber in der .bib-Datei vorkommenden BibTeX-Schlüssel **aus**.

check_all_types

1. Finde alle vorkommenden Literaturtypen in der .bib-Datei.
2. Überprüfe die Validität anhand von Daten zu Literaturtypen einer zuvor importierten Datei.
3. Falls ein Vorkommen nicht valide ist, schließe mit einer Fehlermeldung.

bib_to_dict

1. **Nimm** die Liste von *check_all_keys* **entgegen**.
2. Finde alle einzelnen Einträge der Literaturdatenbank der .bib-Datei und erstelle eine Liste von Tupeln, bestehend aus dem BibTeX-Schlüssel und den einhergehenden Feld-Wert-Zeilen.
3. Erstelle ein Wörterbuch, in dem die BibTeX-Schlüssel auf eine Liste der einzelnen einhergehenden Zeilen aus der Literaturdatenbank verweisen.
4. Bearbeite das Wörterbuch, sodass die BibTeX-Schlüssel auf die einzelnen einhergehenden Felder aus der Literaturdatenbank verweisen und diese wiederum auf die einhergehenden Werte. Nutze dazu die Daten zu Literaturtypen einer zuvor importierten Datei.
5. Identifiziere in den Namenangaben bestimmter Felder den jeweiligen Vor- und Nachnahmen und lass die Felder auf jeweils auf diese beiden seperaten Ebenen verweisen.
6. Entferne alle BibTeX-Schlüssel-Einträge aus dem Wörterbuch, deren BibTeX-Schlüssel nicht in der .txt-Datei vorkommen (mittels der Liste von *check_all_keys*)
7. Erstelle für möglicherweise fehlende Namenangaben der Felder *author* und fehelnde Jahresangaben der Felder *year* Pseudoangaben.
8. **Gib** das Wörterbuch **aus**.

check_required_fields

1. **Nimm** das Wörterbuch von *bib_to_dict* **entgegen**.
2. Erstelle eine Liste von Tupeln, bestehend aus den zusammengehörenden BibTeX-Schlüsseln und Literaturtypen.
3. Prüfe (mittels des Wörterbuchs von *bib_to_dict*), ob alle benötigten Felder des jeweiligen Literaturtyps gegeben sind. Entferne hierfür ggf. zurvor generierte *author*- bzw. *year*-Felder mit Pseudoangaben.
4. Falls ein benötigtes Feld nicht gegeben ist, prüfe anhand von Daten zu Literaturtypen einer zuvor importierten Datei, ob es ggf. durch eine alternativ mögliche Angabe (z.B im Falle *author/editor*) gegeben ist. Falls nicht, schließe mit einer Fehlermeldung.
5. Erstelle ein Wörterbuch aus der Liste von Tupeln, bestehend aus den zusammengehörenden BibTeX-Schlüsseln und Literaturtypen und **gib** dieses **aus**.

transfer

1. **Nimm** das Wörterbuch von *bib_to_dict* **entgegen**.

2. Erstelle eine Liste von allen Anfangspositionen der Zitationsbefehle in der .txt-Datei.
3. Erstelle eine Liste von allen Endpositionen der Zitationsbefehle in der .txt-Datei.
4. Erstelle eine Liste von Tupeln, bestehend aus den zusammengehörigen Anfangs- und Endpositionen der Zitationsbefehle in der .txt-Datei.
5. Transferiere in einer Schleife den Inhalt vom Anfang der .txt-Datei (bzw. von der Endposition des vorigen Zitationsbefehls) bis zur jeweiligen Anfangsposition des Zitationsbefehls in die Ausgabedatei. Anstelle des Zitationsbefehls schreibe (mittels des Wörterbuchs von *bib_to_dict*) eine Autor-Jahr-Zitationsangabe in die Ausgabedatei. Transferiere den Inhalt. Wenn kein Zitationsbefehl folgt, transferiere den restlichen Inhalt der .txt-Datei in die Ausgabedatei.

bibliography

1. **Nimm** das jeweilige Wörterbuch von *bib_to_dict* und *check_required_fields* **entgegen**.
2. Frag, wie das Literaturverzeichnis in der Ausgabedatei betitelt werden soll.
3. Füge an die Ausgabedatei den gewünschten Titel des Literaturverzeichnis an.
4. Erstelle eine Liste von Tupeln, bestehend aus den zusammengehörenden Nachnamen und BibTeX-Schlüsseln.
5. Ordne die Tupel in der Liste alphabetisch (in Bezug auf die Nachnamen) an.
6. Erstelle für in der alphabetischen Reihenfolge für jeden BibTeX-Schlüssel einen Eintrag im Literaturverzeichnis mittels des Wörterbuchs von *bib_to_dict*. Prüfe dafür mit dem Wörterbuch von *check_required_fields* (bestehend aus den zusammengehörenden BibTeX-Schlüsseln und Literaturtypen) den jeweiligen Literaturtyp und berücksichtige dies bei der Erstellung des Eintrags.
7. Optional: Füge eine Signatur des Programms an.

1.1.3 Beispiele

a) Fließtext Zur beispielhaften Darstellung der Funktion des Programms habe ich eine .txt-Datei namens *xmp.txt* folgenden Inhalts erstellt:

*Habe nun, ach! Philosophie, Juristerei und Medizin, und leider auch Theologie durchaus studiert, mit heißem Bemühn [?]. Da steh ich nun, ich armer Tor! Und bin so klug als wie zuvor [?, vgl.]]**Book42**; heiße Magister, heiße Doktor gar [?, 42] und ziehe schon an die zehen Jahr herauf, herab und quer und krumm meine Schüler an der Nase herum [?, vgl.]42]**Conference42** - und sehe, daß wir nichts wissen können! Das will mir schier das Herz verbrennen [?].*

*Zwar bin ich gescheiter als all die Laffen, Doktoren, Magister, Schreiber und Pfaffen [?]; mich plagen keine Skrupel noch Zweifel [?, vgl.]]**Inproceedings42**, fürchte mich weder vor Hölle noch Teufel [?, 42] - dafür ist mir auch alle Freud entrissen [?, vgl.]42]**Mastersthesis42**, bilde mir nicht ein, was Rechts zu wissen, bilde mir nicht ein, ich könnte was lehren, die Menschen zu bessern und zu bekehren [?].*

*Auch hab ich weder Gut noch Geld, noch Ehr und Herrlichkeit der Welt [?, vgl.]42f]**Phdthesis42**; es möchte kein Hund so länger leben! Drum hab ich mich der Magie ergeben [?, vgl.]42ff]**Proceedings42**, ob mir durch Geistes Kraft und Mund nicht manch Geheimnis würde kund; daß ich nicht mehr mit saurem Schweiß zu sagen brauche, was ich nicht weiß [?, vgl.]42]**Techreport42**; daß ich erkenne, was die Welt im Innersten zusammenhält, schau alle Wirkenskraft und Samen, und tu nicht mehr in Worten kramen [?].*

- [?] -

Sie umfasst 15 Zitationsbefehle (hier fett markiert), die auf eine dazugehörigen Literaturdatenbank in der gleichnamigen .bib-Datei *(xmp.bib)* verweisen und alle 14 möglichen Literaturtypen umfassen. Die Zitationsbefehle weisen vier verschiedene Formen auf: 1. [**?**] 1. \cite[**Suffix**]{Schlüssel} 1. \cite[**Präfix**][]{Schlüssel} 1. \cite[**Präfix**][**Suffix**]{Schlüssel}

Diese Formen der Zitationsbefehle werden durch das Programm differenziert behandelt und ersetzt:

1. (Autor Jahr)
2. (Autor Jahr: **Suffix**)
3. (**Präfix** Autor Jahr)
4. (**Präfix** Autor Jahr: **Suffix**)

Dementsprechend hat der obige Inhalt in der Ausgabedatei folgende Form:

*Habe nun, ach! Philosophie, Juristerei und Medizin, und leider auch Theologie durchaus studiert, mit heißem Bemühn **(Arendt 2000)**. Da steh ich nun, ich armer Tor! Und bin so klug als wie zuvor **(vgl. Borchelt 2001)**; heiße Magister, heiße Doktor gar **(n.a. n.d.: 42)** und ziehe schon an die zehen Jahr herauf, herab und quer und krumm meine Schüler an der Nase herum **(vgl. Conert 2003: 42)** - und sehe, daß wir nichts wissen können! Das will mir schier das Herz verbrennen **(Ingham 2004)**.*

*Zwar bin ich gescheiter als all die Laffen, Doktoren, Magister, Schreiber und Pfaffen **(Inselmann 2005)**; mich plagen keine Skrupel noch Zweifel **(vgl. Ipkendanz 2006)**, fürchte mich weder vor Hölle noch Teufel **(n.a. n.d.: 42)** - dafür ist mir auch alle Freud entrissen **(vgl. Massbaum 2008: 42)**, bilde mir nicht ein, was Rechts zu wissen, bilde mir nicht ein, ich könnte was lehren, die Menschen zu bessern und zu bekehren **(n.a. n.d.)**.*

*Auch hab ich weder Gut noch Geld, noch Ehr und Herrlichkeit der Welt **(vgl. Pohl 2010: 42f)**; es möchte kein Hund so länger leben! Drum hab ich mich der Magie ergeben **(vgl. n.a. 2011: 42ff)**, ob mir durch Geistes Kraft und Mund nicht manch Geheimnis würde kund; daß ich nicht mehr mit saurem Schweiß zu sagen brauche, was ich nicht weiß **(vgl. Teckner 2012: 42)**; daß ich erkenne, was die Welt im Innersten zusammenhält, schau alle Wirkenskraft und Samen, und tu nicht mehr in Worten kramen **(Unnerstall n.d.)**.*

*- **(Goethe 2000)** -*

b) Literaturverzeichnis Darüber hinaus ist dem Inhalt in der Ausgabedatei ein Literaturverzeichnis angefügt, welches unter dem gewünschten Titel des Verzeichnisses wie folgt aussieht:

Arendt, Anna (2000): The title of the article. In: The name of the journal 4.

Borchelt, Ben; ed. n.a. (2001): The title of the book. The name of the publisher.

Conert, Connie (2003): The title of the booklet. In: The title of the book.

Goethe, Johann Wolfgang von (2000): Faust: Der Tragödie erster Teil.

Ingham, Ingo; ed. n.a. (2004): The title of the inbook, chapter 8. The name of the publisher.

Inselmann, Ina (2005): The title of the incollection. In: The name of the publisher, The title of the book.

Ipkendanz, Immanuel (2006): The title of the inproceedings. In: The title of the book.

Massbaum, Mara (2008): The title of the mastersthesis. Master's thesis, The school of the thesis.

Pohl, Phillip (2010): The title of the phdthesis. PhD thesis, The school of the thesis.

Teckner, Tina (2012): The title of the techreport. The institution that published.

Unnerstall, Uwe: The title of the unpublished. An optional note.

n.a. (n.d.): The title of the work.

n.a. (n.d.): The title of the inproceedings.

n.a. (n.d.): n.a..

n.a.; ed. n.a. (2011): The title of the proceedings.

Hierbei ist jeder Eintrag in Bezug auf den jeweiligen Literaturtyp, und teilweise bezüglich der gegebenen Felder, kontextsensitiv erstellt. Wie im Falle der Zitationsangabe ist die Angabe des Autors und des Jahres obligatorisch im Literaturverzeichnis. Dass auffallend häufig die generierten Pseudoangaben "n.a." *(no author / not applicable)* bzw. "n.d." *(no date)* erscheinen, ist der Tatsache geschuldet, dass in der beispielhaften .bib-Datei lediglich die Felder angegeben sind, welche der Definition nach erforderlich sind. In Fällen, in denen ein Editor anstatt eines Autors angegeben werden kann, wird dieser im Literaturverzeichnis nach dem Kürzel "e.d." auf den Autor folgend genannt.

c) weitere Reaktionsweisen Zur Darstellung weiterer Reaktionsweisen des Programms habe ich zusätzliche .bib-Dateien erstellt, die im Ordner *xmp* zu finden sind: 1. **additional_author.bib** (zusätzliche Angabe eines Autors) 1. **author_editor_missing.bib** (fehlende erforderliche Angabe von Autor/Editor) 1. **author_missing.bib** (fehlende Angabe eines Autors) 1. **chapter_or_pages_missing.bib** (fehlende Angabe von Kapitel/Seiten) 1. **editor.bib** (zusätzliche Angabe eines Editors) 1. **entry_missing.bib** (fehlende Angabe eines BibTeX-Eintrags) 1. **multiple_authors.bib** (Autorenschaft mehrerer Personen) 1. **no_comma_author.bib** (Angabe des Autorennames ohne Kommaschreibweise) 1. **no_comma_editor.bib** (Angabe des Editorennames ohne Kommaschreibweise) 1. **reoccurrence.bib** (mehrfaches Auftauchen eines BibTeX-Schlüssels) 1. **upper.bib** (Literaturtypen und Felder in Großschreibung) 1. **whitespace.bib** (zusätzlicher willkürlicher Leerraum) 1. **wrong_type.bib** (fehlerhafte Angabe eines Literaturtyps)

Die Reaktionsweisen des Programms lassen sich in vier Gruppen einteilen:

kein Unterschied: 1. no_comma_author.bib 1. upper.bib 1. whitespace.bib

erwünschter Unterschied: 1. additional_author.bib 1. editor.bib

unerwünschter Unterschied: 1. multiple_authors.bib 1. no_comma_editor.bib

Fehlermeldung: 1. author_editor_missing.bib 1. author_missing.bib 1. chapter_or_pages_missing.bib 1. entry_missing.bib 1. reoccurrence.bib 1. wrong_type.bib

d) Ausführung Darüber hinaus weist das Programm – unabhängig der eingelesenen Dateien – in der Benutzung verschiedene Reaktionsweisen auf. Zu Beginn wird der Nutzer über Eingabemöglichkeiten informiert. Er kann (1.) die Eingabetaste drücken, (2.) einen *help*- oder (3.) *quit*-Befehl eingeben. 1. Im ersten Fall fährt das Programm ohne Umwege fort. 1. Bei der Eingabe von *help* wird ein kurzer Informationstext eingeblendet und der Nutzer muss im nächsten Schritt

zum Fortfahren die Eingabetaste drücken. 1. Bei der alternativen Eingabe von *quit* erscheint eine erneute Abfrage, ob das Programm wirklich geschlossen werden soll. Bei einer positiven Antwort ist das Programm beendet. Eine negative Antwort führt zum Neustart. (Diese Verhaltensweise ist während aller Eingaben im Rahmen der Initalisierung möglich und identisch.)

Anschließend wird der Nutzer um die Eingabe eines Pfades zur .txt-Datei gebeten. Gibt dieser einen Pfad an, der nicht valide ist, so wird eine Fehlermeldung ausgegeben und eine erneute Eingabe gefordert. Ein analoges Verhalten weist das Programm bei der anschließenden Pfadeingabe zur .bib-Datei auf. Darauf folgend wird der Nutzer gefragt, ob die Ausgabedatei im aktuellen Arbeitsverzeichnis erstellt werden soll. Eine positive Antwort führt unmittelbar weiter, während eine negative Antwort die Eingabe eines Pfades fordert. Im nächsten Schritt wird nach einem Namen der Ausgabedatei gefragt, der um die Endung .txt ergänzt wird, falls diese nicht explizit angegeben wurde. Abschließend folgt die Frage, ob das Literaturverzeichnis mit der vorgegebenen Überschrift betitelt werden soll. Eine positive Antwort führt zur erfolgreichen Beendigung des Programms, während die Negation zu einer abschließenden Abfrage eines alternativen Titels führt.

Nun eine beispielhafte Ausführung des Programms. Es erstellt in einem neuen Ordner namens *out* die Ausgabedatei namens *xmp_output.txt*. Das Literaturverzeichnis wird anstatt mit "Bibliography" mit *Literaturverzeichnis* betitelt:

[1]:
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""This module contains the class BibTeX."""
import io
import os
import re
import sys
import time
from datetime import date

from data.entry_types import entry_types
from data.entry_types import types
from data.entry_types import fields

no_author = "n.a."
no_title = "n.a."
no_year = "n.d."


class BibTeX(object):
    """Take .txt and .bib file and return output file.

    .txt file includes references and the .bib file constitutes the
    database of all reference-list entries. Output file (.txt) includes
    author-year citations and reference list.
    """

    def __init__(self, txt=None, bib=None, output=None):
```

```
        """Initialize an instance of the class.

        txt -- user's .txt file (default None)
        bib --  user's .bib file (default None)
        output -- path to output file (default None)
        """
        welcome = ("----------------------------"
                   "\nBib.tXt (c) {} Max Harder.\n"
                   "----------------------------".format(date.today().year))
        help = "help"
        quit = "quit"
        introduction = ("\nPress Enter to continue or type \'{}\' for a brief "
                        "description of Bib.tXt. You can always use \'{}\' to "
                        "end the program: ".format(help, quit))
        error = "Error. Document name or path does not end with \'.{}\'."
        ioerror = "Error. The document name or path is not valid."
        path = "Enter relative path for desired .{} file: /"
        print(welcome)
        enter = raw_input(introduction)
        if enter == help:
            try:
                # directory path of this file:
                dir_path = os.path.dirname(os.path.realpath(__file__))
                with io.open(os.path.join(dir_path, 'data/help.txt'),
                             encoding="utf-8") as file:
                    information = file.read()
            except (IOError, NameError):
                with io.open(os.path.abspath('data/help.txt'),
                             encoding="utf-8") as file:
                    information = file.read()
            print(information)
            raw_input("Press Enter to continue: ")
        elif enter == quit:
            return
        print("Your current working directory (cwd) is:")
        cwd = os.getcwd()
        print(cwd)
        while not txt:
            try:
                txt_rel_path = raw_input(path.format("txt"))
                if txt_rel_path == quit:
                    return
                if txt_rel_path.endswith(".txt"):
                    txt_abs_path = os.path.join(cwd, txt_rel_path)
                    with io.open(txt_abs_path, encoding="utf-8") as file:
                        txt = file.read()
                else:
```

```
                print(error.format("txt"))
            except IOError:
                print(ioerror)

        while not bib:
            try:
                bib_rel_path = raw_input(path.format("bib"))
                if bib_rel_path == quit:
                    return
                if bib_rel_path.endswith(".bib"):
                    bib_abs_path = os.path.join(cwd, bib_rel_path)
                    with io.open(bib_abs_path, encoding="utf-8") as file:
                        bib = file.read()
                else:
                    print(error.format("bib"))
            except IOError:
                print(ioerror)

        while not output:
            try:
                path_question = raw_input("Create output file in cwd "
                                          "(y/[n])?: ")
                if path_question == quit:
                    return
                elif path_question == "y":
                    out_name = raw_input("Enter name of output file: ")
                    if out_name.endswith('.txt'):
                        out_full_path = os.path.join(cwd, out_name)
                    else:
                        out_full_path = os.path.join(cwd, out_name+".txt")
                else:
                    out_rel_path = raw_input("Enter relative path "
                                             "for output file: /")
                    if out_rel_path == quit:
                        return
                    out_abs_path = os.path.join(cwd, out_rel_path)
                    if not os.path.exists(out_abs_path):
                        os.makedirs(out_abs_path)
                    out_name = raw_input("Enter name of output file: ")
                    if out_name == quit:
                        return
                    if out_name.endswith('.txt'):
                        out_full_path = os.path.join(out_abs_path, out_name)
                    else:
                        out_full_path = os.path.join(out_abs_path,
                                                     out_name + ".txt")
                output_file = open(out_full_path, "w+")
```

```
                output_file.close()
                output = out_full_path
            except (OSError, IOError):
                print(ioerror)

            self.txt = txt
            self.bib = bib
            self.output = output

    def check_all_keys(self):
        """Check all occurrences of BibTeX keys.

        Return list of keys occurring in .bib but not in .txt file. Exit if
        multiple occurrences of key in .bib file exist or if key in .txt
        file is not specified in .bib file.
        """
        re_keys = r"\\cite(?:\[[^\]]*\])?(?:\[[^\]]*\])?{(\w+)}"
        all_keys_txt = re.findall(re_keys, self.txt)
        all_keys_bib = re.findall(r"@\w+\{(\w+),", self.bib)
        key_not_in_txt = [key for key in all_keys_bib if
                          key not in all_keys_txt]
        for txt_key in all_keys_txt:
            if all_keys_bib.count(txt_key) > 1:
                sys.exit("Error: Reoccurrence of the BibTeX key \'{}\'"
                         " detected. Please revise entered .bib file."
                         .format(txt_key))
            if txt_key not in all_keys_bib:
                sys.exit("Error: \'{}\' is not a specified BibTeX key."
                         " Please revise entered .txt or .bib file."
                         .format(txt_key))
        return key_not_in_txt

    def check_all_types(self):
        """Check all entry types in .bib file. Exit if type is not valid."""
        all_types = re.findall(r"@(\w+)\{", self.bib)
        for mytype in all_types:
            if mytype.lower() not in types:
                sys.exit("Error: \'{}\' is not a valid entry type."
                         " Please revise entered .bib file.".format(mytype))

    def bib_to_dict(self, key_not_in_txt):
        """Convert data of .bib file into dict and return dict.

        Three-layer dict: keys, fields, values.
        """
        re_bib = r"@\w+\{"+"(\w+)"+",([\w\s={}\/\.:,;-]+)(?=(?:@\w+\{)|\Z)"
        basis_dict, field_value_dict = {}, {}
```

```
        # tuple_list: [(key, field=value,\n etc.) etc.]
        tuple_list = [match.group(1, 2) for match in
                      re.finditer(re_bib, self.bib, re.UNICODE)]
        # key_val_tuple: (key, field=value,\n …)
        key_list = [key_val_tuple[0] for key_val_tuple in tuple_list]
        for key_val_tuple in tuple_list:
            basis_dict[key_val_tuple[0]] = key_val_tuple[1].splitlines()
        for key in key_list:
            # unsplit_data: field=value
            for unsplit_data in basis_dict[key]:
                # strip whitespace from the beginning of the string:
                unsplit_data = unsplit_data.lstrip()
                if any([unsplit_data.lower()
                        .startswith(field) for field in fields]):
                    split_data = re.split(r"\s+=\s+", unsplit_data)
                    final_value = re.search(r"{*([\w\s\/\.:,;-]+)}*",
                                            split_data[1], re.UNICODE).group(1)
                    # remove comma after digit and full stop after title etc.:
                    if final_value.endswith((",", ".")):
                        final_value = final_value[:-1]
                    field_value_dict[split_data[0].lower()] = final_value
            basis_dict[key] = {}
            basis_dict[key].update(field_value_dict)
            field_value_dict = {}
            if "author" in basis_dict[key]:
                if "," in basis_dict[key]["author"]:
                    split_name = basis_dict[key]["author"].split(",")
                    surname = split_name[0].strip()
                    forename = split_name[1].strip()
                    basis_dict[key]["author"] = {"surname": surname}
                    basis_dict[key]["author"].update({"forename": forename})
                else:
                    split_name = basis_dict[key]["author"].split()
                    surname = split_name[-1].strip()
                    forename = " ".join(split_name[:-1]).strip()
                    basis_dict[key]["author"] = {"surname": surname}
                    basis_dict[key]["author"].update({"forename": forename})
        # remove keys not occurring in .txt file
        for ghost_key in key_not_in_txt:
            basis_dict.pop(ghost_key)
        for key in basis_dict.keys():
            basis_dict[key]["author"] = basis_dict[key].get(
                                        "author", {"surname": no_author,
                                                   "forename": ""})
            basis_dict[key]["year"] = basis_dict[key].get("year", no_year)
        return basis_dict
```

```
    def check_required_fields(self, basis_dict):
        """Check if required fields are given in .bib file.

        Return dict of keys and their entry types. Exit if required field is
        not given.
        """
        field_error = ("Error: \'{}\' is a required field for \'{}\'. "
                       "Please revise entered .bib file.")
        field_error_2 = ("Error: \'{}\' or \'{}\' is a required field for "
                         "\'{}\'. Please revise entered .bib file.")
        key_type_tuples = [match.group(2, 1) for match
                           in re.finditer(r"@(\w+)\{(\w+),", self.bib)]
        for mytuple in key_type_tuples:
            for key in basis_dict.keys():
                if key in mytuple:
                    given = [field.lower() for field in basis_dict[key].keys()]
                    # remove generated entries for 'author':
                    if basis_dict[key]["author"]["surname"] == no_author:
                        given.remove("author")
                    for needed in entry_types[mytuple[1].lower()]["required"]:
                        split_fields = [element for element
                                        in needed.split("/")
                                        if re.findall(r"\w+/\w+", needed)]
                        if needed not in given:
                            if re.findall(r"\w+/\w+", needed):
                                if (split_fields[0] not in given
                                   and split_fields[1] not in given):
                                    sys.exit(field_error_2
                                             .format(split_fields[0],
                                                     split_fields[1],
                                                     mytuple[1]))
                            else:
                                sys.exit(field_error
                                         .format(needed, mytuple[1]))
        key_type_dict = dict(key_type_tuples)
        return key_type_dict

    def transfer(self, basis_dict):
        """Transfer (modified) content of .txt to output file.

        Transfer content and convert references into author-year citations.
        """
        all_varieties = r"(\\cite(?:\[([^\]]*)\])?(?:\[([^\]]*)\])?{(\w+)})"
        page_only = r"(\\cite(?:\[([^\]]*)\]){(\w+)})"
        # quotes: [(u'\\cite[x][y]{z}', u'x', u'y', u'z') etc.]
        quotes = re.findall(all_varieties, self.txt)
        cite_start = ([match.start() for match in re.finditer
```

```
                    (r"\\cite(?:\[[^\]]*\]){0,2}{\w+}", self.txt)])
        cite_end = ([match.end() for match in re.finditer
                     (r"\\cite(?:\[[^\]]*\]){0,2}{\w+}", self.txt)])
        # cite_position: [(1st, 1st), (2nd, 2nd) etc.]
        cite_position = list(zip(cite_start, cite_end))
        start, count, num = 0, 1, 0
        for position in cite_position:
            with io.open(self.output, mode="a", encoding="utf-8") as file:
                file.write(self.txt[start:position[0]])
                # page(+suffix):
                if (re.match(page_only, quotes[num][0])
                   and quotes[num][1] and not quotes[num][2]):
                    file.write(u"({} {}: {})".format(
                               basis_dict[quotes[num][3]]["author"]["surname"],
                               basis_dict[quotes[num][3]]["year"],
                               quotes[num][1]))
                # prefix:
                elif quotes[num][1] and not quotes[num][2]:
                    file.write(u"({} {} {})".format(
                               quotes[num][1],
                               basis_dict[quotes[num][3]]["author"]["surname"],
                               basis_dict[quotes[num][3]]["year"]))
                # prefix and page(+suffix):
                elif quotes[num][1] and quotes[num][2]:
                    file.write(u"({} {} {}: {})".format(
                               quotes[num][1],
                               basis_dict[quotes[num][3]]["author"]["surname"],
                               basis_dict[quotes[num][3]]["year"],
                               quotes[num][2]))
                # nothing:
                elif not quotes[num][1] and not quotes[num][2]:
                    file.write(u"({} {})".format(
                               basis_dict[quotes[num][3]]["author"]["surname"],
                               basis_dict[quotes[num][3]]["year"]))
                if count == len(cite_position):
                    file.write(self.txt[position[1]:])
            start = position[1]
            count += 1
            num += 1

    def bibliography(self, data, key_type_dict):
        """Append bibliography to output file."""
        entitle = raw_input("Entitle reference list \'Bibliography\' (y/[n])?")
        if entitle == "y":
            title = u"Bibliography"
            superscription = u"\n{}\n{}".format(title, len(title)*u"=")
        else:
```

```
            title = ""
            while not title:
                try:
                    title = (raw_input("Enter title of reference list: ")
                             .decode("utf-8"))
                except UnicodeDecodeError:
                    print("Error. Please use \"UTF-8\" or avoid special "
                          "characters.")
            superscription = u"\n{}\n{}".format(title, len(title)*u"=")
        signature = u"\n\nGenerated with Bib.tXt (c) by Max Harder. {}."
        with io.open(self.output, mode="a", encoding="utf-8") as file:
            file.write(superscription)
            surname_key_tuples = [(data[element]["author"]["surname"],
                                   element) for element in data]
            surname_key_tuples.sort()
            for tuple in surname_key_tuples:
                # article
                if key_type_dict[tuple[1]].lower() == "article":
                    file.write(u"\n{}, {} ({}): {}. In: {} {}.".format(
                               data[tuple[1]]["author"]["surname"],
                               data[tuple[1]]["author"]["forename"],
                               data[tuple[1]]["year"],
                               data[tuple[1]]["title"],
                               data[tuple[1]]["journal"],
                               data[tuple[1]]["volume"]))
                # book
                elif key_type_dict[tuple[1]].lower() == "book":
                    if data[tuple[1]]["author"]["surname"] != no_author:
                        file.write(u"\n{}, {}; ed. {} ({}): {}. {}.".format(
                                   data[tuple[1]]["author"]["surname"],
                                   data[tuple[1]]["author"]["forename"],
                                   data[tuple[1]].get("editor", no_author),
                                   data[tuple[1]]["year"],
                                   data[tuple[1]]["title"],
                                   data[tuple[1]]["publisher"]))
                    else:
                        file.write(u"\n{}; ed. {} ({}): {}. {}.".format(
                                   data[tuple[1]]["author"]["surname"],
                                   data[tuple[1]].get("editor", no_author),
                                   data[tuple[1]]["year"],
                                   data[tuple[1]]["title"],
                                   data[tuple[1]]["publisher"]))
                # booklet
                elif key_type_dict[tuple[1]].lower() == "booklet":
                    if data[tuple[1]]["author"]["surname"] != no_author:
                        file.write(u"\n{}, {} ({}): {}.".format(
                                   data[tuple[1]]["author"]["surname"],
```

```
                        data[tuple[1]]["author"]["forename"],
                        data[tuple[1]]["year"],
                        data[tuple[1]]["title"]))
            else:
                file.write(u"\n{} ({}): {}.".format(
                        data[tuple[1]]["author"]["surname"],
                        data[tuple[1]]["year"],
                        data[tuple[1]]["title"]))
        # conference
        elif key_type_dict[tuple[1]].lower() == "conference":
            file.write(u"\n{}, {} ({}): {}. In: {}.".format(
                    data[tuple[1]]["author"]["surname"],
                    data[tuple[1]]["author"]["forename"],
                    data[tuple[1]]["year"],
                    data[tuple[1]]["title"],
                    data[tuple[1]]["booktitle"]))
        # inbook
        elif key_type_dict[tuple[1]].lower() == "inbook":
            if data[tuple[1]]["author"]["surname"] != no_author:
                if ("chapter" in data[tuple[1]]
                   and "pages" in data[tuple[1]]):
                    file.write(u"\n{}, {}; ed. {} ({}): {}, "
                               "chapter {}, pages {}. {}.".format(
                                data[tuple[1]]["author"]["surname"],
                                data[tuple[1]]["author"]["forename"],
                                data[tuple[1]].get("editor",
                                                   no_author),
                                data[tuple[1]]["year"],
                                data[tuple[1]]["title"],
                                data[tuple[1]]["chapter"],
                                data[tuple[1]]["pages"],
                                data[tuple[1]]["publisher"]))
                if ("chapter" in data[tuple[1]]
                   and "pages" not in data[tuple[1]]):
                    file.write(u"\n{}, {}; ed. {} ({}): {}, "
                               "chapter {}. {}.".format(
                                data[tuple[1]]["author"]["surname"],
                                data[tuple[1]]["author"]["forename"],
                                data[tuple[1]].get("editor",
                                                   no_author),
                                data[tuple[1]]["year"],
                                data[tuple[1]]["title"],
                                data[tuple[1]]["chapter"],
                                data[tuple[1]]["publisher"]))
                if ("chapter" not in data[tuple[1]]
                   and "pages" in data[tuple[1]]):
                    file.write(u"\n{}, {}; ed. {} ({}): {}, "
```

```
                                "pages {}. {}.".format(
                                 data[tuple[1]]["author"]["surname"],
                                 data[tuple[1]]["author"]["forename"],
                                 data[tuple[1]].get("editor",
                                                    no_author),
                                 data[tuple[1]]["year"],
                                 data[tuple[1]]["title"],
                                 data[tuple[1]]["pages"],
                                 data[tuple[1]]["publisher"]))
                    else:
                        if ("chapter" in data[tuple[1]]
                           and "pages" in data[tuple[1]]):
                            file.write(u"\n{}; ed. {} ({}): {}, "
                                "chapter {}, pages {}. {}.".format(
                                 data[tuple[1]]["author"]["surname"],
                                 data[tuple[1]].get("editor",
                                                    no_author),
                                 data[tuple[1]]["year"],
                                 data[tuple[1]]["title"],
                                 data[tuple[1]]["chapter"],
                                 data[tuple[1]]["pages"],
                                 data[tuple[1]]["publisher"]))
                        if ("chapter" in data[tuple[1]]
                           and "pages" not in data[tuple[1]]):
                            file.write(u"\n{}; ed. {} ({}): {}, "
                                "chapter {}. {}.".format(
                                 data[tuple[1]]["author"]["surname"],
                                 data[tuple[1]].get("editor",
                                                    no_author),
                                 data[tuple[1]]["year"],
                                 data[tuple[1]]["title"],
                                 data[tuple[1]]["chapter"],
                                 data[tuple[1]]["publisher"]))
                        if ("chapter" not in data[tuple[1]]
                           and "pages" in data[tuple[1]]):
                            file.write(u"\n{}; ed. {} ({}): {}, "
                                "pages {}. {}.".format(
                                 data[tuple[1]]["author"]["surname"],
                                 data[tuple[1]].get("editor",
                                                    no_author),
                                 data[tuple[1]]["year"],
                                 data[tuple[1]]["title"],
                                 data[tuple[1]]["pages"],
                                 data[tuple[1]]["publisher"]))
                # incollection
                elif key_type_dict[tuple[1]].lower() == "incollection":
                    file.write(u"\n{}, {} ({}): {}. In: {}, {}.".format(
```

```
                    data[tuple[1]]["author"]["surname"],
                    data[tuple[1]]["author"]["forename"],
                    data[tuple[1]]["year"],
                    data[tuple[1]]["title"],
                    data[tuple[1]]["publisher"],
                    data[tuple[1]]["booktitle"]))
            # inproceedings
            elif key_type_dict[tuple[1]].lower() == "inproceedings":
                file.write(u"\n{}, {} ({}): {}. In: {}.".format(
                    data[tuple[1]]["author"]["surname"],
                    data[tuple[1]]["author"]["forename"],
                    data[tuple[1]]["year"],
                    data[tuple[1]]["title"],
                    data[tuple[1]]["booktitle"]))
            # manual
            elif key_type_dict[tuple[1]].lower() == "manual":
                if data[tuple[1]]["author"]["surname"] != no_author:
                    file.write(u"\n{}, {} ({}): {}.".format(
                            data[tuple[1]]["author"]["surname"],
                            data[tuple[1]]["author"]["forename"],
                            data[tuple[1]]["year"],
                            data[tuple[1]]["title"]))
                else:
                    file.write(u"\n{} ({}): {}.".format(
                            data[tuple[1]]["author"]["surname"],
                            data[tuple[1]]["year"],
                            data[tuple[1]]["title"]))
            # masterthesis
            elif key_type_dict[tuple[1]].lower() == "masterthesis":
                file.write(u"\n{}, {} ({}): {}. "
                    "Master's thesis, {}.".format(
                     data[tuple[1]]["author"]["surname"],
                     data[tuple[1]]["author"]["forename"],
                     data[tuple[1]]["year"],
                     data[tuple[1]]["title"],
                     data[tuple[1]]["school"]))
            # misc
            elif key_type_dict[tuple[1]].lower() == "misc":
                if data[tuple[1]]["author"]["surname"] != no_author:
                    file.write(u"\n{}, {} ({}): {}.".format(
                            data[tuple[1]]["author"]["surname"],
                            data[tuple[1]]["author"]["forename"],
                            data[tuple[1]]["year"],
                            data[tuple[1]].get("title", no_title)))
                else:
                    file.write(u"\n{} ({}): {}.".format(
                            data[tuple[1]]["author"]["surname"],
```

```
                                data[tuple[1]]["year"],
                                data[tuple[1]].get("title", no_title)))
                # phdthesis
                elif key_type_dict[tuple[1]].lower() == "phdthesis":
                    file.write(u"\n{}, {} ({}): {}. PhD thesis, {}.".format(
                               data[tuple[1]]["author"]["surname"],
                               data[tuple[1]]["author"]["forename"],
                               data[tuple[1]]["year"],
                               data[tuple[1]]["title"],
                               data[tuple[1]]["school"]))
                # proceedings
                elif key_type_dict[tuple[1]].lower() == "proceedings":
                    if data[tuple[1]]["author"]["surname"] != no_author:
                        file.write(u"\n{}, {}; ed. {} ({}): {}.".format(
                                data[tuple[1]]["author"]["surname"],
                                data[tuple[1]]["author"]["forename"],
                                data[tuple[1]].get("editor", no_author),
                                data[tuple[1]]["year"],
                                data[tuple[1]]["title"]))
                    else:
                        file.write(u"\n{}; ed. {} ({}): {}.".format(
                                data[tuple[1]]["author"]["surname"],
                                data[tuple[1]].get("editor", no_author),
                                data[tuple[1]]["year"],
                                data[tuple[1]]["title"]))
                # techreport
                elif key_type_dict[tuple[1]].lower() == "techreport":
                    file.write(u"\n{}, {} ({}): {}. {}.".format(
                               data[tuple[1]]["author"]["surname"],
                               data[tuple[1]]["author"]["forename"],
                               data[tuple[1]]["year"],
                               data[tuple[1]]["title"],
                               data[tuple[1]]["institution"]))
                # unpublished
                elif key_type_dict[tuple[1]].lower() == "unpublished":
                    file.write(u"\n{}, {}: {}. {}.".format(
                               data[tuple[1]]["author"]["surname"],
                               data[tuple[1]]["author"]["forename"],
                               data[tuple[1]]["title"],
                               data[tuple[1]]["note"]))
            file.write(signature.format(date.fromtimestamp(time.time())))
        print("Output successfully created.\n"
              "-----------------------------")


def pipe():
    """Combine all methods of BibTeX()."""
```

```
    example = BibTeX()
    key_not_in_txt = example.check_all_keys()
    example.check_all_types()
    basis_dict = example.bib_to_dict(key_not_in_txt)
    key_type_dict = example.check_required_fields(basis_dict)
    example.transfer(basis_dict)
    example.bibliography(basis_dict, key_type_dict)


def run():
    """Run combined methods of BibTeX() as final program."""
    state = True
    while state is True:
        try:
            pipe()
            state = False
        except AttributeError:
            close = raw_input("Close program (y/[n])? ")
            if close == "y":
                sys.exit()


if __name__ == '__main__':
    # code below is only executed when the module is run directly
    run()
```

```
----------------------------
Bib.tXt (c) 2018 Max Harder.
----------------------------

Press Enter to continue or type 'help' for a brief description of Bib.tXt. You
can always use 'quit' to end the program: help

Bib.tXt takes as input a .txt file (including references) and a .bib file that
constitutes the database of all reference-list entries. It chooses from the .bib
file those entries specified by the .txt file (that is, those given by its \cite
commands), and creates as output a .txt file in which all references are
converted into author-year citations. Aditionally, Bib.tXt will use the input to
produce a reference list that completes the output file.
For additional information use the README file (README.TXT).

Press Enter to continue:
Your current working directory (cwd) is:
/home/marx/Dokumente/Studium/WS 17
18/Texttechnologie/Programmierung/Projekt/2919411
Enter relative path for desired .txt file: /xmp/xmp.txt
Enter relative path for desired .bib file: /xmp/xmp.bib
```

```
Create output file in cwd (y/[n])?: n
Enter relative path for output file: /out
Enter name of output file: xmp_output
Entitle reference list 'Bibliography' (y/[n])?n
Enter title of reference list: Literaturverzeichnis
Output successfully created.
----------------------------
```

1.1.4 Bestehende Probleme

Zwei weiterhin bestehende Probleme des Programms lassen sich mit den .bib-Dateien veranschaulichen, welche einen erwünschten Unterschied hervorrufen: (1.) *multiple_authors.bib* und (2.) *no_comma_editor.bib*. 1. Die Autorenschaft mehrerer Personen in der Datei multiple_authors.bib, gekennzeichnet durch die Verkettung mit 'and' oder '&', ist ein typischer Fall, mit dem das Programm in der vorliegenden Version nicht ausreichend umgehen kann. Das Problem wird im Literaturverzeichnis sichtbar, in dem Einträge wie - *"Arendt, Anna **AND Goethe** (2000): […]."*

ausgegeben werden, anstatt des gewünschten - *"Arendt, Anna **und Goethe, Johann Wolfgang von** (2000): […]."*

oder einfacher: - *"Arendt, Anna **et al.** (2000): […]."*

Eine mögliche Lösung des Problems würde darin bestehen, die Funktion *bib_to_dict* erkennen zu lassen, ob die Schlüsselzeichen(ketten) 'and' bzw. '&' im Wert des Feldes *author* vorkommen. Falls dies der Fall sein sollte, könnte lediglich der erste Name berücksichtigt und um ein 'et al.' ergänzt werden.

2. Die Angabe des Editorennames ohne Kommaschreibweise in der Datei no_comma_editor.bib hat in der Ausgabedatei scheinbar keine unerwünschte Folgen, da im Literaturverzeichnis wohlformatierte Einträge zu finden sind wie

   - *n.a.; ed. Ben Borchelt (2001): […].*

   Dies ist allerdings lediglich auf die Tatsache zurückzuführen, dass das Programm die Namenangaben außerhalb der Autorenfelder nicht näher analysiert und aufspaltet, sondern schlicht die vollständige Zeichenkette als einzelnen Namen wiedergibt. Dies hat zur Folge, dass die Namen von Editoren etc. im Literaturverzeichnis so ausgegeben werden, wie sie in der .bib-Datei eingegeben werden. Dieser Fehler im Programm ist in der Anwendung zu umgehen, indem die Angaben in der .bib-Datei den eigenen Wünschen entsprechend angepasst werden. Um den Fehler im Programm zu beheben, müssten bei allen Feldern bei denen Personennamen angegeben werden können, ähnliche Analysemethoden wie im Falle der Autorenfelder angewendet werden.

Ein weiteres Problem liegt ebenfalls in der Erzeugung des Literaturverzeichnisses verborgen, da durch das Programm – neben Autor, Jahr und Titel – lediglich die erforderlichen Felder des jeweiligen Literaturtyps des spezifischen Eintrags in das Literaturverzeichnis übernommen werden. Durch diese Arbeitsweise können relevante Informationen aus der Literaturdatenbank verloren gehen. Zur Behebung dieses Problems könnte geprüft werden, welche optionalen Felder des jeweiligen Eintrags gegeben sind und eine entsprechende Formatierung in der Ausgabe des Eintrags im Literaturverzeichnis gewählt werden. Dies hätte allerdings eine beträchtliche Verlängerung des Codes zur Folge,

solange kein zusammenfassender Algorithmus für diesen Prozess erstellt wird, der mit allen Literaturtypen umgehen kann und diesbezüglich nicht differenzieren muss.
