**DuSchuldestMirBot**

**---**

**Prototyp einer Chatbot-basierten Erinnerung zum Einlösen von Schulden aller Art**

Inhaltsverzeichnis

[1.Einführung 1](#_Toc35181304)

[2. Technische Voraussetzungen/Umsetzung 2](#_Toc35181305)

[3. Programmablauf/Benutzung 2](#_Toc35181306)

[4. Funktionen (technische Umsetzung) 3](#_Toc35181307)

[4. 1. Registrierung und Start-GUI 3](#_Toc35181308)

[4. 2. Speicherung der Schulden 5](#_Toc35181309)

[4. 3. Schulden eintragen 8](#_Toc35181310)

[4. 4. Schulden begleichen 10](#_Toc35181311)

[4. 5. Bestätigungs-/Ablehnungsmechanismus 12](#_Toc35181312)

[4. 6. Erinnerungsmechanismus/Timer 15](#_Toc35181313)

# 1.Einführung

Dieses Dokument dient der technischen Dokumentation des „DuSchuldestMirBot“, ein Telegram-Bot, entwickelt vom Kurs TIM19 der DHBW RV.

Der „DuSchuldestMirBot“ soll ein Prototyp darstellen, der zu Erinnerung von Schulden jeglicher Art entwickelt ist. Anwender können durch Chatinteraktionen mit einem Bot (also einem Programm) Schulden von anderen Anwendern anfordern oder Schulden begleichen (Getränke, Fahrten, Döner, etc.). Hierbei wird der Schuldner von einer beschriebenen Schuld, inklusive der Einlösefrist, informiert. Der Schuldner kann die Schuld nun annehmen oder ablehnen.

Im Falle der Annahme wird der Schuldner regelmäßig an das Einlösen der Schuld erinnert, die er dann auch im Chat als beglichen eintragen kann.

Die Kriterien der Entwicklung unseres Chatbots waren nach Absprache wie folgt: Der Chatbot sollte ein Telegram-Bot sein, der mit den Anwendern interagiert und chattet. Die Entwicklung bzw. das Skript soll mit Python3 unter der Einbindung der python-telegram-bot-Bibliothek erfolgen. Die Schulden/User werden als JSON abgespeichert.

# 2. Technische Voraussetzungen/Umsetzung

Um den Telegram-Bot in Betrieb zu nehmen, wird natürlich zu allererst ein Telegram-Account benötigt. Wenn die Accounterstellung abgeschlossen ist, sucht man mit der Suchfunktion von Telegram nach dem „Botfather“. Diesen kann man mit „/start“ starten. Mit dem Befehl /newbot erstellt man nun seinen eigenen Bot und folgt den Anweisungen des „Botfather“. Bei erfolgreicher Erstellung eines Bots erhält man einen Token, mit dem man die http API ansprechen kann.

Dieser Token wird nun an der Stelle „###“ in der Python-Datei des eigentlichen Bots eingefügt. Weitere Rahmenbedingungen sind zudem die Installation von Python (Version 3) und die Installation der Pakete „python-telegram-bot“ und „telegram“ (z.B. mit pip).

Nun ist der Bot startklar. Nach Starten des Bots kann man anfangen sich mit dem Bot über den Telegramchat zu unterhalten. Jeder User kann nun den Bot mit dem vom User erstellten Nicknamen mit der Telegram-Suchfunktion finden. Durch Öffnen des Chats kann man sich dann registrieren.

Nach erfolgreicher Registrierung beim Bot kann man nun anderen Usern, die sich bei dem Bot registriert haben, Schulden zuweisen bzw. kann selbst Schulden zugewiesen bekommen.

# 3. Programmablauf/Benutzung

Wie schon in den technischen Vorrausetzungen erklärt, kann man den Bot mit der Telegram-Suchfunktion über den vom Ersteller festgelegten Nicknamen suchen.

Beim Öffnen des Chats muss sich der User bei Erstbenutzung zuerst mit dem Befehl /start registrieren (wenn er das auch möchte) und sich somit in der Datenbank (bzw. in die User-JSON-File) des Bots eintragen.

Nach erfolgreicher Registrierung erscheint in dem Chat zwischen User und Bot eine Auswahl an Befehlen, die von hier aus aufgeführt werden können. Diese können jederzeit zur Steuerung des Bots verwendet werden.

Mit dem Befehl /schuld kann man einem User, der auch beim Bot registriert ist, eine Schuld zuweisen. Hierfür wählt man die Art, Menge und das Datum, zu der die Schuld spätestens beglichen sein soll. Der Schuldner bekommt daraufhin eine Bestätigungsbenachrichtigung, in der er die Schuld bestätigen/ablehnen kann. Wenn die Schuld bestätigt wurde, bekommen Schuldner und Gläubiger regelmäßig eine Benachrichtigung, die einen an die Schuld erinnern soll.

Mit dem Befehl /ichBekomme kann sich der User alle Schulden anzeigen lassen, die dem User zustehen.

Mit dem Befehl /ichSchulde zeigt sich der User alle offenen Schulden an, die er verschuldet. Hier kann durch Auswahl einer Schuld die Schuld beglichen werden. Der Gläubiger bekommt in diesem Fall eine Benachrichtigung, um zu bestätigen, ob die Schuld wirklich beglichen wurde. Erst nachdem es bestätigt wurde, gilt die Schuld als beglichen und wird auch so gespeichert.

# 4. Funktionen (technische Umsetzung)

Hier werden die einzelnen Funktionen des Programms technisch und logisch erläutert.

## 4. 1. Registrierung und Start-GUI

start(update, context)

Beim Start des Bots wird automatisch abgefragt, ob der User bereits registriert ist. Falls „Nein“, wird der User gefragt, ob eine Registrierung gewünscht ist. Dadurch wird eine ungewollte Registrierung verhindert und es bleibt dem User selbst überlassen, ob er sich registrieren möchte. Dies kann über Buttons entschieden werden. Wenn er sich registrieren möchte, erfolgt eine Weiterleitung ins Startmenü und der Nutzer wird in der Datenbank gespeichert.

#region registration of users

def start(update, context):

    #Username that addresses the bot is written on "name"

    name = update.message.from\_user.first\_name

    chat\_ID = str(update.effective\_message.chat\_id)

    registrated = db.user\_exists(chat\_ID)

    if(registrated):

        update.message.reply\_text('Hey ' + name + '!' + '\nWillkommen zurück!'+'\U0001F60F')

        startMenu(update, context) #startMenue in startMenu!!!!!

    else:

        #Bot answer: Greets the user by name

        update.message.reply\_text('Hey ' + name + '!' + '\nWillkommen bei Deinem lokalen Anbieter für Schuldeneintreibung!'+

        '\U0001F60F')

Falls der registrierte User den Chatbot neu startet, erhält er keine neue Abfrage zur Registrierung, sondern wird mit Namen begrüßt und direkt ins Startmenü weitergeleitet.

Die Abfragen werden mittels einem *InlineKeyboard* durchgeführt und die Reaktion in der Methode *„button“* weiterverarbeitet.

keyboardYN =  [[InlineKeyboardButton("\U0001F44D", callback\_data='yes'), #keyboardJN in keyboardYN geändert!!!

                       InlineKeyboardButton("\U0001F44E", callback\_data='no')]]

        reply\_markup = InlineKeyboardMarkup(keyboardYN) #create keybard

        update.message.reply\_text('Möchtest Du dich registrieren?', reply\_markup = reply\_markup)

        UPDATER.dispatcher.add\_handler(CallbackQueryHandler(button)) #registration

button(update, context)

Diese Methode wird aufgerufen, wenn bei der Registrierung der Button „Ja“ oder „Nein“ gedrückt wird. Der jeweilige Button speichert beim Auslösen einen für ihn spezifischen Wert in die Variable *callback\_data,* welche wiederum eine Reaktion auslöst mit entsprechender Methode.

Ein Beispiel hierfür wäre: ‚Yes‘ wird für „Ja“ in der *callback\_data* abgespeichert und leitet entsprechende Reaktion und Methode weiter ein, wenn der Button „Ja“ gedrückt wird.

def button(update, context):

    query = update.callback\_query

    #if 'yes' the user will be registratet and is forwarded to the start menu

    if query.data == 'yes':

        startMenu(update, context)

        #edited

        chat\_ID = str(query.message.chat\_id)

        user\_name = str(query.from\_user.username)

        db.add\_user(chat\_ID, user\_name)

        # if no , no registration

    elif query.data == 'no':

        cancel(update, context)

cancel(update, context)

Falls der User keine Registrierung wünscht, wird die Methode „*cancel“* aufgerufen. Hier wird der *ConversationHandler* beendet und der User bekommt den Hinweis durch /start den Prozess neu zu starten. Somit wird ihm die Wahl gelassen, sich nochmal umzuentscheiden.

def cancel(update, context):

    query = update.callback\_query

    bot = context.bot

    #Closing text

    bot.edit\_message\_text(

        chat\_id=query.message.chat\_id,

        message\_id=query.message.message\_id,

        text="Schade!" + "\nFalls Du es dir anders überlegst, kannst du mit /start den Prozess neu starten." + "\U0001F47C"

    )

    return ConversationHandler.END

startMenu(update, context)

Das Hauptmenü kann mittels Befehle wie: /schuld, /ichBekomme und /ichSchulde gesteuert werden. Diese Befehle können jederzeit eingegeben werden. Die Methode erzeugt eine Übersicht, bzw. eine Liste der Befehle, sodass der Nutzer weiß, welche Möglichkeiten er für Steuerbefehle hat. Die Reaktion auf einen Befehl wird in den entsprechenden Methoden weiterverarbeitet.

def startMenu(update, context):

    query = update.callback\_query

    bot = context.bot

    #import Username + ChatId

    chat\_ID = str(query.from\_user.id)

    user\_name = str(query.from\_user.username)

    bot.edit\_message\_text(

        #generate startmenu

        chat\_id = query.message.chat\_id,

        message\_id = query.message.message\_id,

        text = "Bitte wähle dein Anliegen aus:")

    context.bot.send\_message(query.from\_user.id, text="Klicke auf \u27A1 /schuld um Schulden einzutragen...")

    context.bot.send\_message(query.from\_user.id, text="Klicke auf \u27A1 /ichSchulde um einzusehen,

wem du was schuldest...")

    context.bot.send\_message(query.from\_user.id, text="Klicke auf \u27A1 /ichBekomme um einzusehen,

was dir wer schuldet...")

## 4. 2. Speicherung der Schulden

Um die Speicherung der Daten zu ermöglichen wurden drei Klassen erstellt:

User:

Die User Klasse stellt einen Verbund der Attribute eines Benutzers dar.

Ein User Objekt besitzt folgende Attribute:

chat\_id: Chat ID des Chats zwischen Benutzer und Bot

name: Username des Benutzers

Die Klasse besitzt folgende Funktionen:

to\_dict: Gibt die Attribute des User Objekts in einem Dictionary zurück

from\_dict: Wandelt ein gegebenes Dictionary in ein User Objekt um und gibt das User Objekt anschließend zurück

Debt:

Die Debt Klasse stellt ein Verbund der einzelnen Attribute eines Schuldenprozesses dar.

Ein Debt Objekt besitzt folgende Attribute:

debt\_id: ID des Schuldenprozesses, als UUID Zahl

creditor: Chat ID des Gläubigers

category: Kategorie der Schuld, z. B. Mittagessen

amount: Wert der Schuld in Euro

deadline: Datum bis wann die Schulden bezahlt werden sollen

debtor: Chat ID des Schuldners

is\_accepted: Zeigt, ob der Schuldner die Schuld akzeptiert hat

is\_paid: Zeig, ob der Schuldner die Schuld beglichen hat

Die Klasse besitzt folgende Funktionen:

to\_dict: Gibt die Attribute des Debt Objekts in einem Dictionary zurück

from\_dict: Wandelt ein gegebenes Dictionary in ein Debt Objekt um und gibt das Debt Objekt anschließend zurück

Database:

Die Database Klasse interagiert mit der JSON Datei. Also werden mithilfe dieser Klasse die einzelnen User und Debt Objekte gespeichert beziehungsweise ausgelesen.

Die Klasse besitzt folgende Attribute:

path\_to\_json: Dateipfad der JSON Datei

users: Liste der User Objekte

debts: Liste der Schuld Objekte

Die Klasse besitzt folgende Funktionen:

init\_json: Lädt die Daten der JSON Datei und initialisiert die User- und Debtliste

update\_json: Speichert die aktuellen Listen in der JSON Datei

user\_exist: Überprüft anhand der Chat ID, ob ein bestimmter User bereits in der Datenbank existiert

add\_user: Fügt einen neuen User zur Datenbank hinzu

add\_debt: Fügt einen neuen Schuldenprozess zur Datenbank hinzu

get\_open\_debts: Gibt alle offenen Schulden eines Users zurück

get\_open\_claim: Gibt alle offenen Forderungen eines Users zurück

get\_user\_by\_chat\_id: Gibt das passende User Objekt zu einer Chat ID zurück

get\_debt\_by\_debt\_id: Gibt das passende Schuld Objekt zu einer Debt ID zurück

set\_accepted: Setzt das is\_accepted Attribut eines Debt Objektes

set\_paid: Setzt das is\_paid Attribut eines Debt Objektes

Im Folgenden wird auf das verwendete Dateiformat JSON eingegangen:

JSON steht für „JavaScript Object Notation“ und ist ein für den Mensch und die Maschinen leicht zu verstehendes Textformat zum Austausch von Daten. Es basiert auf dem JavaScript-Standard ECMA-262 3rd Edition, welcher im Dezember 1999 verabschiedet wurde. Das Format besteht aus Key / Value Paaren, welche ineinander verschachtelt werden können. Beispielsweise kann der Value eines Keys aus beliebig vielen weiteren Key / Value Paaren bestehen. Dies ist aus einigen Programmiersprachen als Dictionary, Listen bzw. Arrays bekannt. Daher ist JSON leicht und ohne großen Aufwand zu implementieren. Die JSON Daten werden in der Datenbank Klasse mithilfe der JSON Python Library verarbeitet.

Verwendung der JSON Library:

Zur Verarbeitung der User- und Schuldobjektdaten, wird auf Funktionen des importierten JSON Moduls zurückgegriffen. Folgende Funktionen wurden verwendet:

Json.loads:

Deserialisiert einen String-Instanz, welche ein JSON-Dokument enthält, in ein Python-Objekt.

Json.dumps:

Serialisiert ein Python-Objekt zu einer JSON formatierten String-Instanz.

Datenmodell:

Das Datenmodell stellt die zu Verarbeitenden bzw. gespeicherten Daten dar. Die Userobjektdaten werden in der Liste / dem Array „users“ gespeichert. Diese Liste beinhaltet alle Attribute eines Users in Form eines Dictionarys. Jeder User wird in einem separaten Dictionary dargestellt. Schuldobjektdaten werden ebenfalls in einer Liste gespeichert, die Liste „debts“. Für jedes Schuldobjekt existiert ein Dictionary in der Liste.

{

"users": [

{

"chat\_id": "chat\_id",

"name": "user\_name"

}

],

"debts": [

{

"debt\_id": "963c5a9c-66ed-11ea-bee7-784f4377cbf0",

"creditor": "creditor\_id",

"category": "Mittagessen",

"amount": 12,

"deadline": "1970.01.01",

"debtor": "debtor\_id",

"is\_accepted": false,

"is\_paid": false

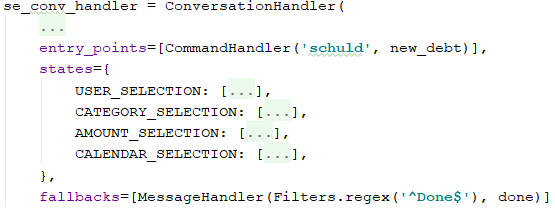
}

]

}

## 4. 3. Schulden eintragen

Nachdem sich ein User erfolgreich registriert hat, hat dieser drei verschiedene Optionen zur Auswahl. Neben den Anzeigeoptionen für eigene Verschuldungen und von anderen noch nicht beglichenen Schulden, lassen sich neue Schulden in Auftrag geben. Die Infos zur neuen Schuld werden nach dem Ausführen von „/schuld“ in einer bestimmten Reihenfolge abgefragt.



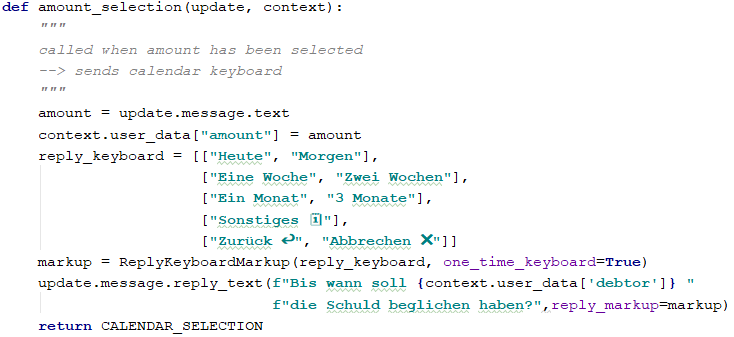
Das zentrale Herzstück dieses Befehls ist der „ConversationHandler“, der jeglichen User-Input verarbeitet. Verschiedene Eingaben sorgen dafür, dass der ConversationHandler bestimmte Methoden je nach Zustand aufruft. Die Zustände sorgen dafür, dass wenn beispielsweise die Eingabe eines Datums erwartet wird, ein Username nicht akzeptiert wird.

Somit läuft der Informationsgewinn für die Schuld nach folgendem Konzept ab:

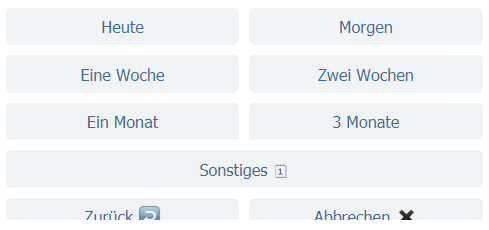
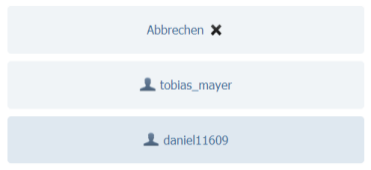
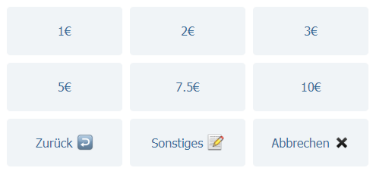
***Auswahl des Schuldners  Kategorie  Anzahl  Zeitpunkt***

Der Input des Users wird je nach Zustand anders verarbeitet. Dafür gibt vorgefertigte Textfilter (*regex*) oder auch nur den „rohen“ Userinput (*text*), die wiederum verschiedene Methoden (z.B. *amount\_selection*) aufrufen.

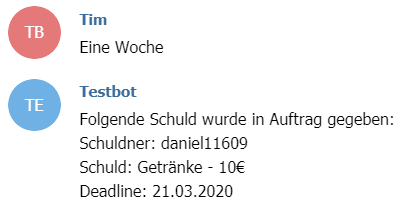
Was bei allen Zuständen jedoch gleich bleibt, sind die Abbruchsfunktionen, wie z.B. das Aufrufen anderer Befehle und die „zurück“ Funktion.



Die Methoden selbst können den Status des Conversation Handlers ändern und senden meist auch Informationen an den User, wie z.B. Custom Keyboards für den Fortlauf.

Über verschiedene Tastaturen, die je nach der zu abfragenden Info passend gestaltet sind, kann der User schnell und komfortabel Eingaben machen.

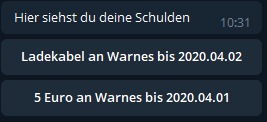
Über einen in Telegram integrierten Knopf, der sich über den individuell für dieses Projekt gestalteten Tastaturen befindet, kann jederzeit zu einer klassischen Tastatur gewechselt werden, um bei den Eingaben alle Freiheiten zu haben (siehe regex/text im Abschnitt ConversationHandler). Zu Beginn wählt man eine der aufgelisteten Personen. Mit dem Wechsel auf die traditionelle Tastatur, kann man Namen auch manuell eingeben, insofern die Person registriert ist und die Eingabe keine Tippfehler enthält. Mit der darauffolgenden Abfrage der Kategorie, lässt sich die Art der Verschuldung angeben. Wie es danach weiter geht, hängt von der Wahl der Kategorie ab. Je nach Art der Verschuldung werden verschiedene Tastaturen nach der Wahl der Kategorie geöffnet, mithilfe deren man den Wert der Verschuldung am besten angeben kann. Zu guter Letzt ist noch die Angabe einer sogenannten „Deadline“ relevant, um alle nötigen Daten zur Erstellung eines neuen Eintrags erfasst zu haben. Neben einigen vorgefertigten Auswahlmöglichkeiten lässt sich hier ebenfalls selber ein Grenzdatum angeben. Nach dem Abschluss der Eintragungen, wird eine Anfrage der Schuld an die zu Beginn gewählte Person geschickt. Zuletzt ist bei diesem Punkt noch zu erwähnen, dass jede funktionell angepasste Tastatur die grundlegenden Funktionen des Abbrechens und des Zurückkehrens zur letzten Wahl zur Verfügung stellt.



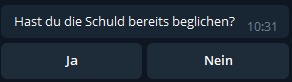
Um die Schuld nun als Sammlung aus Daten einzureichen, gibt es Schnittstellen, die an andere Ecken des gesamten Programms anknüpfen. Die rohen Daten werden zunächst zur Speicherung weitergegeben und erhalten eine ProzessID. Diese wird für den nächsten Schritt benötigt, in dem die Daten zusammen mit der neu erhaltenen ID an die Accept-/Deny-Funktion weitergegeben werden, von der aus die Anfrage für die Schulden verschickt wird. Durch die ID ist die Anfrage nun zuzuordnen und kann zukünftig bei Annahme oder Ablehnung weiterbearbeitet werden.

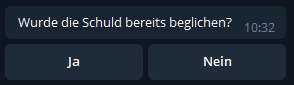
## 4. 4. Schulden begleichen

Dieser Teil des Projekts ist dafür verantwortlich, dass bereits eingestellte Schulden ordnungsgemäß beglichen werden können. Dies kann sowohl von Gläubiger- als auch von Schuldnerseite angestoßen werden. In jedem Fall bedarf es allerdings der Zustimmung der Gegenseite. Jeder Nutzer kann sich sowohl eine Liste seiner Schulden als auch seiner Forderungen ausgeben lassen.

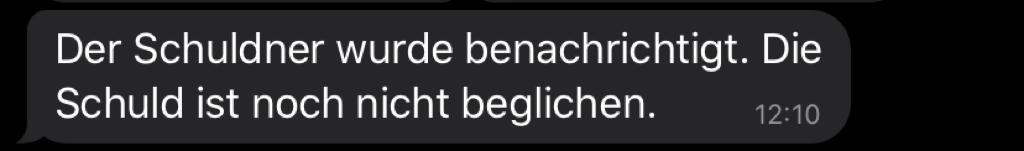
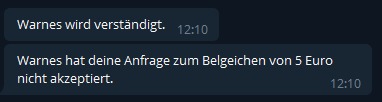
Implementiert wurden zwei Befehle: /ichSchulde und /ichBekomme, welche jeweils vom Hauptmenü aus aufgerufen werden können. Bei beiden Befehlen wurde darauf geachtet dieselbe Struktur im Ablauf des Prozesses beizubehalten.

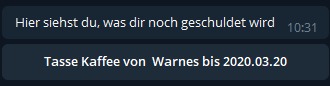
Beim Aufrufen von /ichSchulde wird dem User eine Liste seiner Schulden, welche er an einen anderen Nutzer zurückzuzahlen hat, angezeigt. Diese Liste umfasst folgende Details: Beschreibung der Schuld, Gläubiger, spätester Rückzahlungszeitpunkt.

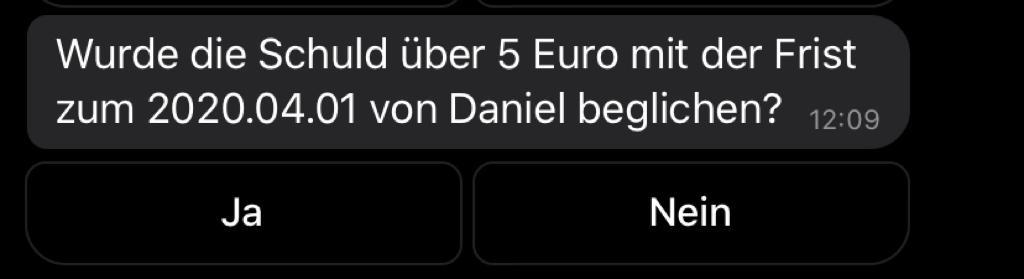
Durch das Anklicken einer in der Liste aufgeführten Schuld wird dem User die Option einer Begleichung angeboten, welche er entweder annehmen oder ablehnen kann. Im Falle einer Ablehnung bekommt er lediglich eine kurze Rückmeldung „Ok“ und die Liste wird geschlossen, anschließend hat er die Möglichkeit beliebig fortzufahren.

Im Falle einer Annahme wird der Gläubiger darüber benachrichtigt, dass der Schuldner seine Schuld begleichen möchte. Der Gläubiger hat nun die Möglichkeit dies zu bestätigen und damit der Begleichung zuzustimmen oder er lehnt die Begleichung der Schuld ab, zum Beispiel dann, wenn er die Rückzahlung nicht erhalten hat.

Sollte der Gläubiger die Rückzahlung bestätigen, ist der Vorgang beendet, die Schuld wird aus der Liste des Schuldners entfernt. Wenn der Schuldner also keine Nachricht nach dem Begleichen der Schuld erhält, dann weiß er, dass er die Schuld erfolgreich beglichen hat. Dies kann er mit dem Befehl „/ichSchulde“ kontrollieren, denn hier taucht diese bereits beglichene Schuld nun nicht mehr auf. Zusätzlich erhält er auch keine weiteren Zahlungserinnerungen mehr.

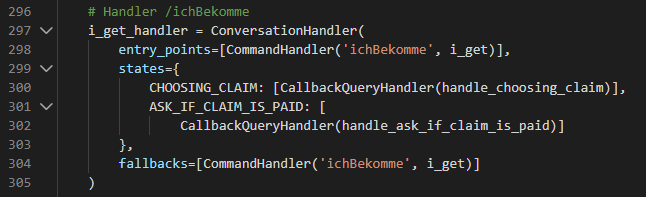
Sollte der Gläubiger die Rückzahlung ablehnen, so erhält der Schuldner eine Nachricht darüber, dass die Rückzahlung nicht akzeptiert wurde und weiß somit, dass die Schuld weiterhin besteht. Somit sind Täuschungsversuche, sowie Missverständnisse über noch offene und bereits beglichene Schulden ausgeschlossen.

Beim Aufrufen von /ichBekomme wird dem User eine Liste mit all seinen Auslagen angezeigt. Auch hier enthält die Liste folgende Details: Beschreibung der Auslage, Schuldner, spätester Rückzahlungstermin.

Beim Auswählen einer in der Liste aufgeführten Auslage gibt es die Möglichkeit das Begleichen dieser Auslage zu bestätigen oder abzulehnen. Somit hat auch der Gläubiger die Möglichkeit eine Auslage als erledigt zu markieren.

Bei einer Ablehnung der Rückzahlung einer Auslage bekommt der Gläubiger die kurze Antwort „Ok“. Damit ist der Prozess beendet, es wird niemand informiert und es werden auch keine Auslagen/Schulden aus der Liste entfernt.

Bei der Bestätigung der Rückzahlung einer Auslage bekommt der Gläubiger die Nachricht darüber, dass die Auslage als beglichen markiert wurde. Der Schuldner erhält keine gesonderte Nachricht. Eine aktuelle Übersicht über die Schulden und Auslagen können sich Gläubiger sowie Schuldner jeder Zeit über die beiden Befehle /ichSchulde und /ichBekomme anzeigen lassen.

Umgesetzt wurde der oben beschriebene Ablauf mit insgesamt drei Handlern, die in Kombination mehrfach benutzt werden. Es werden zwei ConversationHandler benutzt, welche auf die „states“ reagieren, die von einer anderen Methode zurückgegeben werden.

Wie bereits erwähnt werden in Kombination zwei weitere Handler verwendet: Der CommandHandler und der CallbackQueryHandler.

Mit dem CommandHandler machen wir es den Usern möglich, unsere Befehle /ichSchulde und /ichBekomme zu nutzen und dadurch mit dem Bot zu kommunizieren. Dabei greifen wir auf den CallbackQueryHandler zurück.

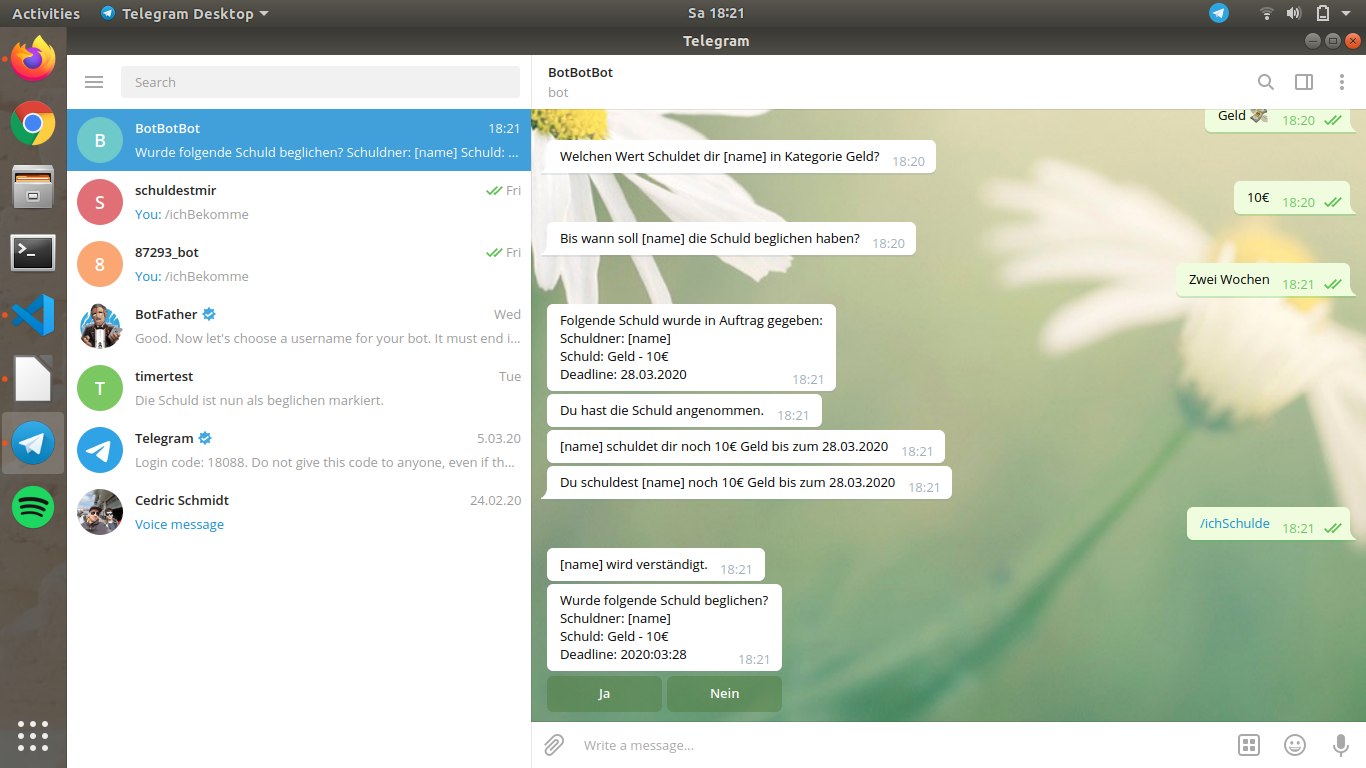
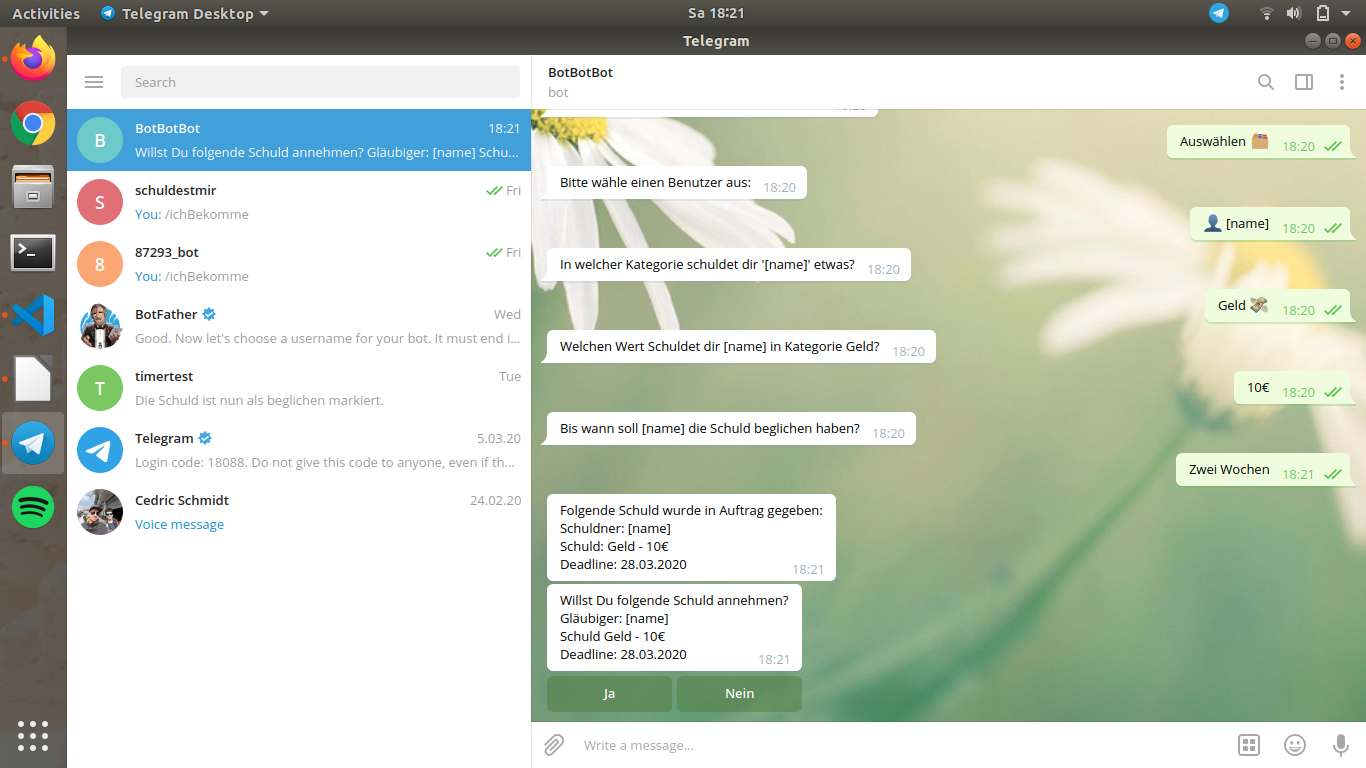
Die Aufgabe des CallbackQueryHandler ist es, dem Bot korrekt wiederzugeben, welcher Button vom User gedrückt wurde. Beispielsweise ob der User „Ja“ oder „Nein“ als Antwort ausgewählt hat oder welche Schuld vom Nutzer ausgewählt wurde. Wir verwenden hierbei eine „debt\_id“ um das zu identifizieren.

Es gibt einen weiteren separaten CallbackQueryHandler, der dazu dient, die Bestätigung des Gläubigers zu bearbeiten. Dies kann nicht im ConversationHandler geschehen, da der Bot mit mehreren Usern kommunizieren muss.

## 4. 5. Bestätigungs-/Ablehnungsmechanismus

Da ein User andere User nicht beliebig mit Schulden behaften können sollte, musste ein Mechanismus eingeführt werden, der den vom Gläubiger angegebenen Schuldner bestätigen lässt, dass er eine Schuld annimmt.

Außerdem muss der Bot die Funktionalität enthalten, dass ein Gläubiger die Begleichung einer Schuld bestätigen kann, nachdem ein Schuldner diese angibt.



Annehmen einer Schuld

Beim Eintragen einer Schuld durchläuft der Nutzer ein Formular, dessen letzter Schritt das Eintragen einer Deadline ist. In derselben Funktion (calendar\_selection), in der das Eintragen dieser Deadline behandelt wird, wird eine Nachricht mit einem Annehm-/ und einem Ablehnbutton an den Schuldner geschickt. Bei Betätigen eines der Buttons wird ein Objekt der Klasse CallbackQuery an den Bot gesendet, dessen data-Attribut ein String mit dem Format „[1 oder 0],[debt\_id]“ ist.

Eine weitere Funktion (handle\_accept\_debt) bestimmt, ob der „Ja“- oder der „Nein“-Button betätigt wurde und prüft dementsprechend die erste Stelle des callback\_query.data-String auf „1“ oder „0“. Wurde „Ja“ gedrückt und folglich ein String, der eine „1“ enthält, an den Bot gesendet, werden drei Aktionen ausgelöst:

Die Nachricht mit dem InlineKeyboard wird editiert, um den Nutzer zu informieren, dass der Vorgang gelungen ist.

Das „accepted“-Attribut der Schuld der Klasse Debt wird auf True gesetzt (set\_paid). Das Debt-Objekt wird dabei über dessen eindeutige ID bestimmt (Database.get\_debt\_by\_debt\_id).

Und der Timer für diese Schuld wird gestartet (start\_timer).

Bei Betätigung des “Nein”-Buttons hingegen wird nicht nur der Schuldner, sondern auch der Gläubiger informiert, dass die Schuld nicht als angenommen markiert wurde. Ein Timer wird nicht gestartet und auch der Wett des „accepted“-Attributs nicht geändert.

Begleichung einer Schuld bestätigen

Bei diesem Teil des Codes, wird die Nachricht zur Bestätigung an den Gläubiger geschickt. Dies passiert innerhalb der letzten Funktion (handle\_ask\_if\_debt\_is\_paid), die vom ConversationHandler i\_owe\_handler aufgerufen wird. Die bei Betätigung eines Buttons mitgeschickten Daten sind hier anders organisiert als die in „Annehmen einer Schuld“ behandelten. Ein Python-dict mit den Schlüsselwerten ‚paid‘ und ‚debt\_id‘ wird als String im Json-Format gespeichert (json.dumps).

Abgefragt wird dieser String als callback\_query.data durch erneute Umformung zu einem dict (json.loads) in der Funktion handle\_accept\_debt\_is\_paid. Diese Funktion setzt in einem ersten Schritt das paid-Attribut der behandelten Schuld auf den Wert, der im dict mit dem Key ‚paid‘ gespeichert wurde, also True oder False.

Außerdem wird bei True die Nachricht wie beim Annehmen einer Schuld editiert und der Timer gestoppt (stop\_timer)

Bei False wird die Nachricht ebenfalls editiert und eine Nachricht an den Schuldner gesendet, dass die Anfrage zur Begleichung abgelehnt wurde. Der Timer wird nicht gestoppt.

Handler-Organisation

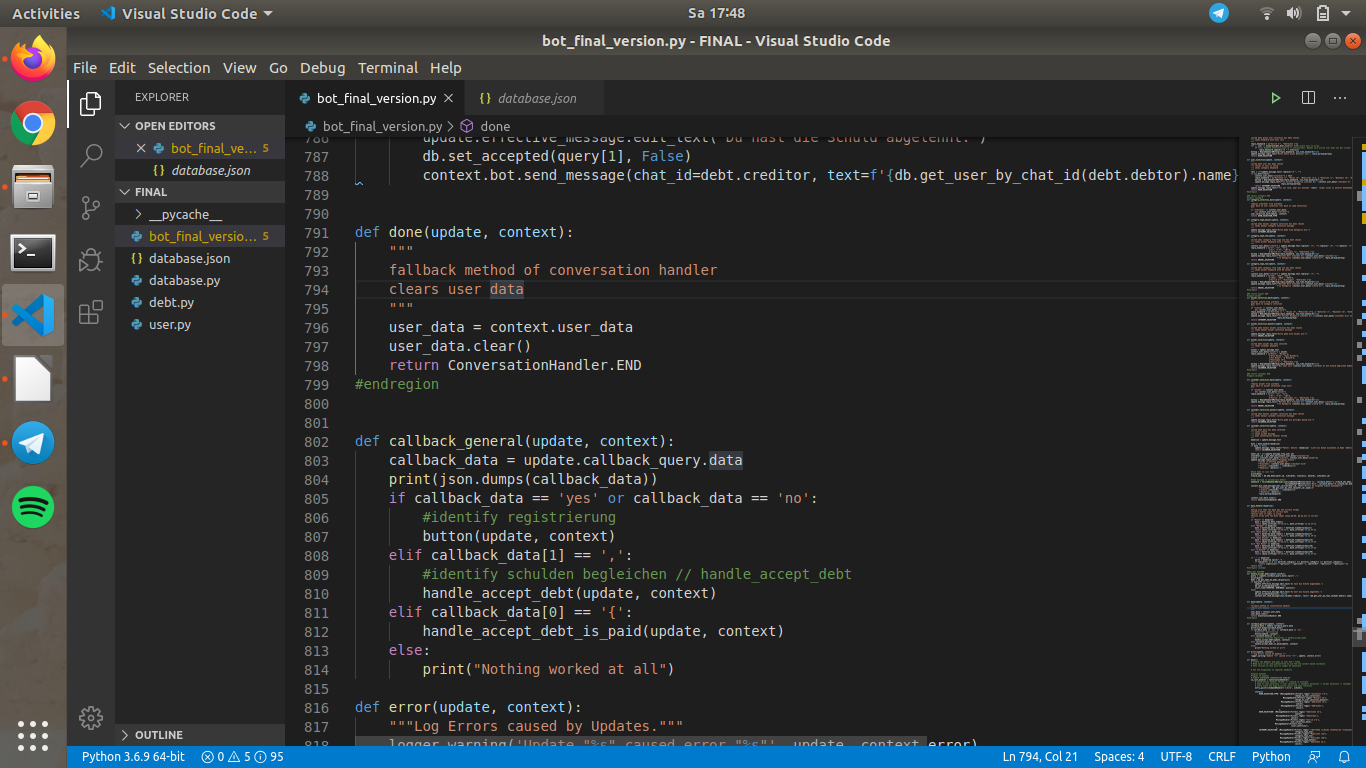
Ein ConversationHandler, wie er für die Vorgänge, eine Schuld einzutragen und eine Schuld zu begleichen, eingesetzt wird, ist dafür ausgelegt die Nachrichten eines einzelnen Nutzers zu behandeln. Deswegen müssen die beschriebenen Funktionen (handle\_accept\_debt, handle\_accept\_debt\_is\_paid) zur Behandlung der CallbackQuerys anderer Nutzer anders aufgerufen werden als mithilfe der ConversationHandler. Sie brauchen ausgelagerte CallbackQueryHandler, um aufgerufen zu werden.

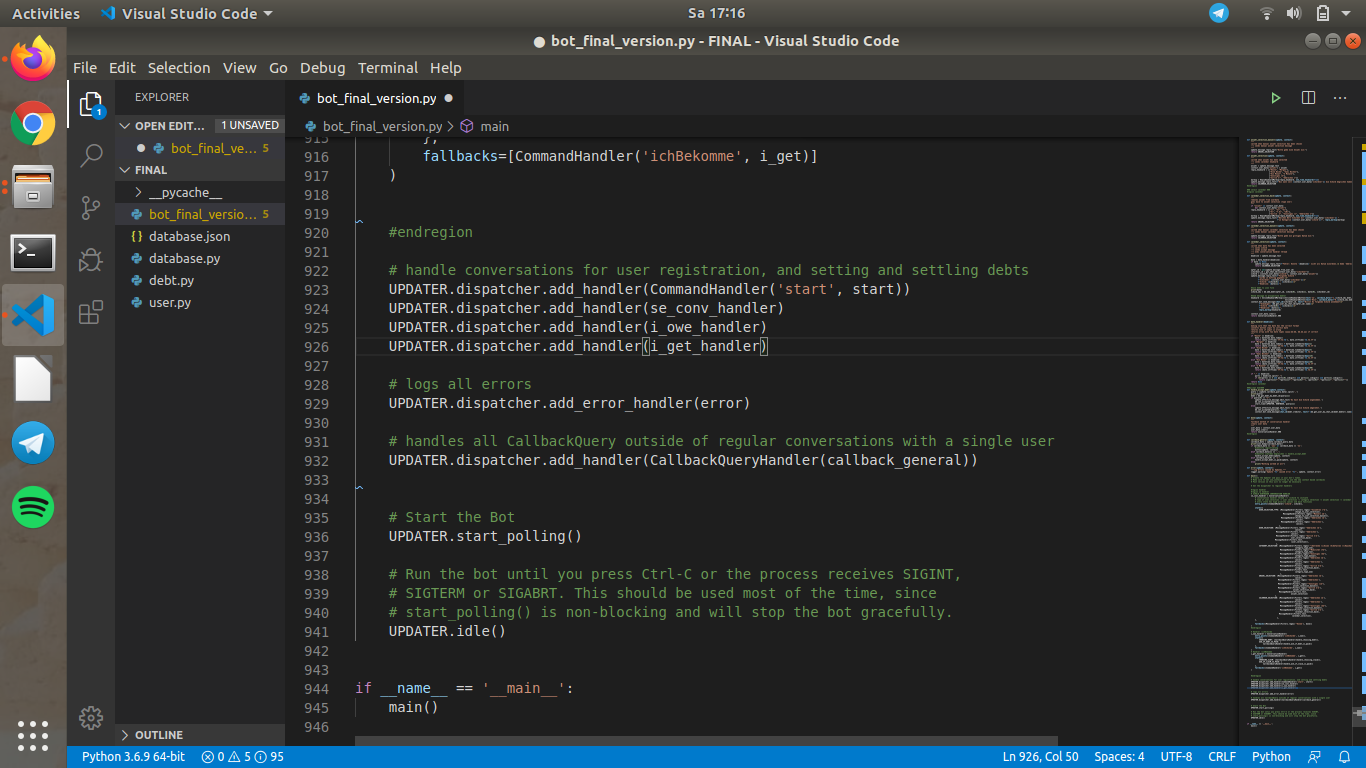
Da beide Annehmefunktionalitäten eines ausgegliederten CallbackQueryHandlers bedürfen, wäre es denkbar, dass der finale Code zwei Objekte dieser Handler-Klasse beinhalten würde.

Dem ist nicht so, weil mehrere CallbackQueryHandler nicht parallel laufen können; insbesondere, weil ein CallbackQueryHandler nicht auf ein CallbackQuery eines bestimmten Nutzers reagiert, sondern auf jedes CallbackQuery-Objekt, welches der Bot empfängt.

Also wird eine weitere Funktion (callback\_general) definiert. Sie wird bei jedem CallbackQuery aufgerufen, welches nicht innerhalb einer bestehenden Konversation auftritt.

Jedes CallbackQuery wird dabei durch einfache if-Abfragen seinem Zweck zugeordnet und die entsprechende Funktion aufgerufen.





Ergänzungen

Das CallbackQuery.data, das in handle\_accept\_debt behandelt wird hat ein anderes Format als das, welches in handle\_accept\_debt\_is\_paid behandelt wird, weil das callback\_data-Attribut der Klasse InlineKeyboardButton eine Größenbegrenzung hat. Diese Größenbegrenzung ließ für handle\_accept\_debt nur eine unzuverlässige Nutzung des eigentlich optimalen json-Formats zu, weswegen ein anderes Format genutzt wurde.

In callback\_general wird außerdem die Funktion button aufgerufen, weil die Registrierung nicht über einen ConversationHandler gelöst wurde.

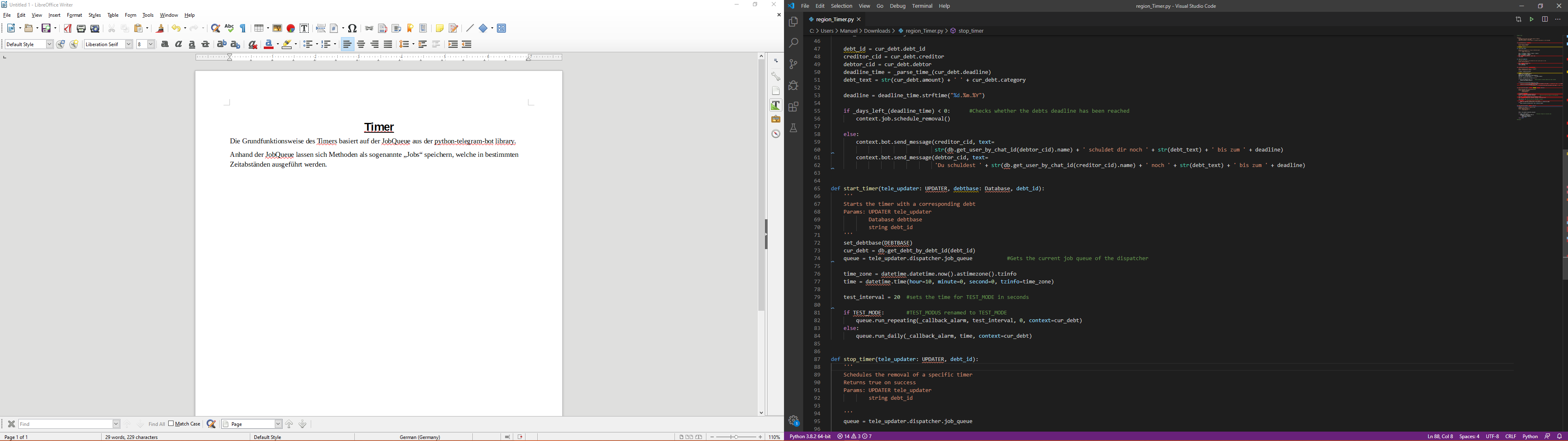
## 4. 6. Erinnerungsmechanismus/Timer

Die Aufgabe des Timers ist, in regelmäßigen Abständen Schuldner und Gläubiger an ihre offenen Schulden zu erinnern.

Die Grundfunktionsweise des Timers basiert auf der *JobQueue* aus der *python-telegram-bot library*.

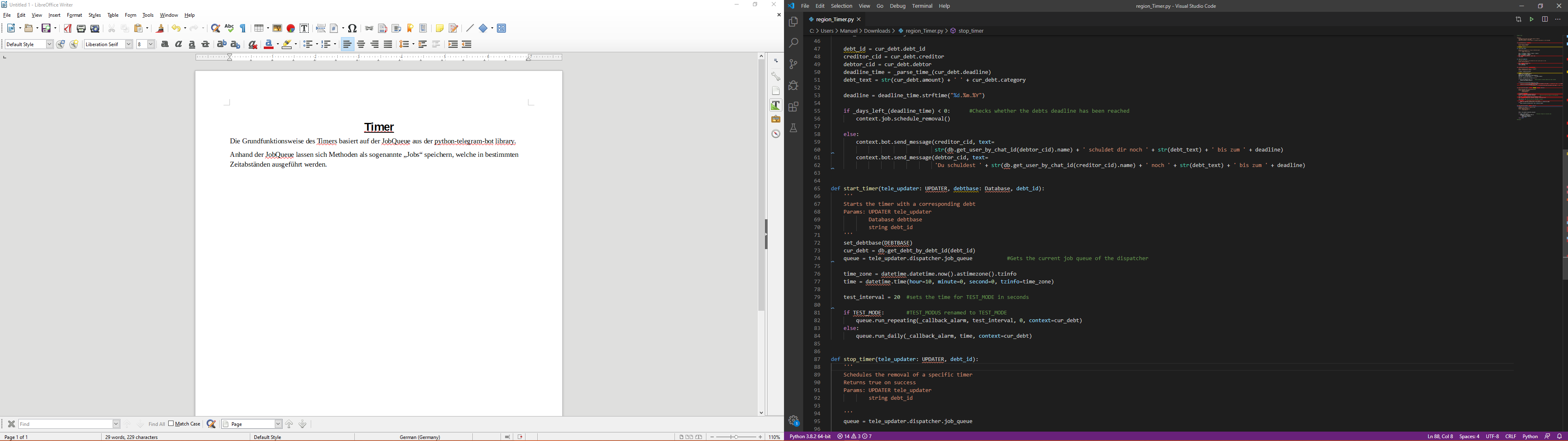
Innerhalb der *start\_timer* – Methode lässt sich die Funktionsweise des *JobQueue* gut darstellen.

Anhand der *JobQueue* lassen sich Methoden als sogenannte *Jobs* speichern, welche in bestimmten Zeitabständen ausgeführt werden. Man kann wie folgt mit der *JobQueue* arbeiten:



Damit erhält man die *JobQueue* des aktuellen *Dispatcher-Objekts*. *tele\_updater* ist hier der zugehörige *Updater*.

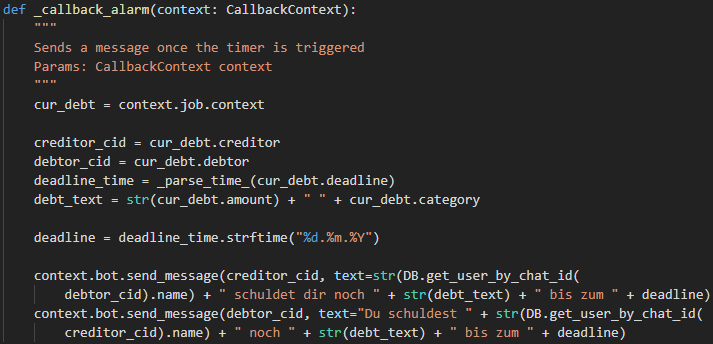
Das starten eines Jobs funktioniert u.A. mit folgendem Befehl:



Hiermit lässt sich der Job *\_callback\_alarm* einmal täglich zu einer festgelegten Zeit *time* ausführen.

Als *context* lässt sich hier ein beliebiges Objekt übergeben.

Unsere \_*callback\_alarm* - Methode kümmert sich um das Benachrichtigen des Schuldners und des Gläubigers.

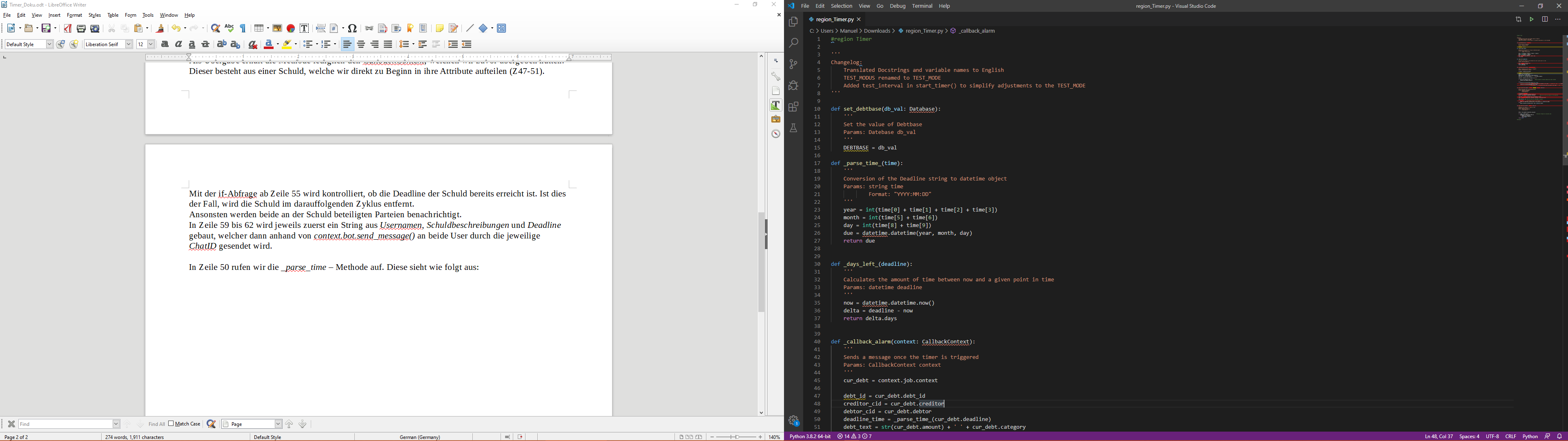


Als Übergabe erhält die Methode lediglich den *CallbackContext*, welchen wir zuvor übergeben hatten.

Dieser besteht aus einer Schuld, welche wir direkt zu Beginn in ihre benötigten Attribute aufteilen.

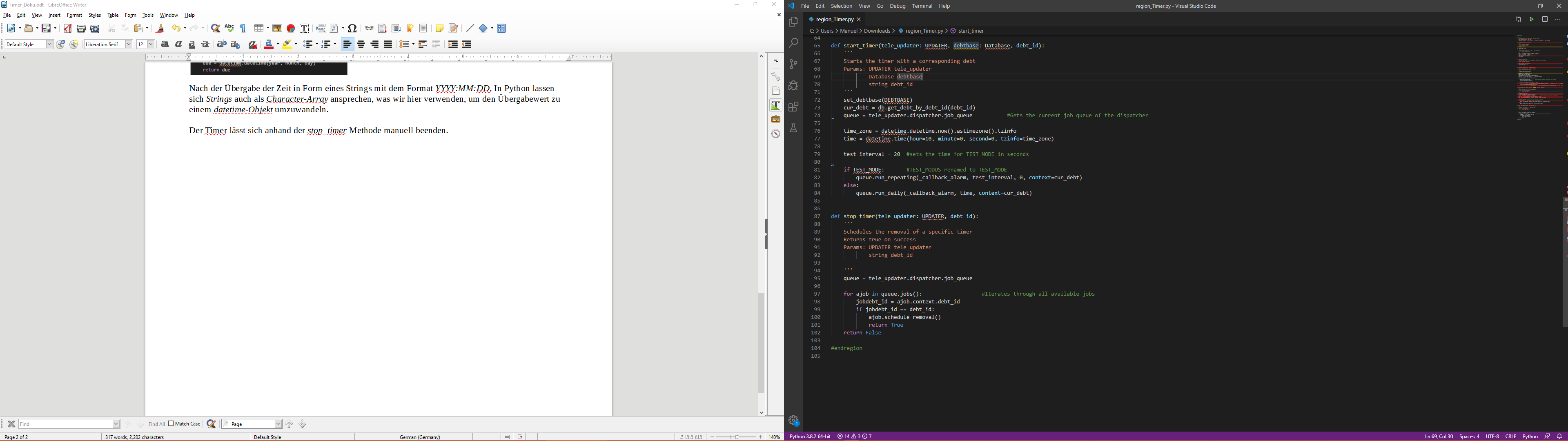
In den unteren vier Zeilen wird jeweils zuerst ein String aus *Usernamen, Schuldbeschreibungen* und *Deadline* gebaut, welcher dann anhand von *context.bot.send\_message()* an beide User durch die jeweilige *ChatID* gesendet wird.

Um deadline\_time festzulegen rufen wir die \_*parse\_time –* Methode auf. Diese sieht wie folgt aus:



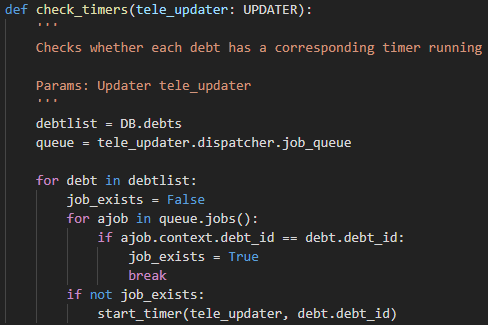
Nach der Übergabe der Zeit in Form eines Strings mit dem Format *YYYY:MM:DD*. In Python lassen sich *Strings* auch als *Character-Array* ansprechen, was wir hier verwenden, um den Übergabewert zu einem *datetime-Objekt* umzuwandeln.

Der Timer lässt sich anhand der *stop\_timer* Methode manuell beenden.



Diese benötigt als Übergabewerte den *Updater* und die *debt\_id* der Schuld, welche entfernt werden soll. Die einzelnen Jobs der *JobQueue* werden in der *for-schleife* durchlaufen und anhand einer *if-Abfrage* wird überprüft ob der zu prüfende Job dieselbe *debt\_id* besitzt. Ist dies der Fall, wird der Job in dem darauffolgenden Zyklus entfernt. Bei Erfolg wird *True* zurückgegeben, bei Misserfolgen wird *False* zurückgegeben.

Nun müssen wir bei einem Neustart des Bots absichern, dass sämtliche Timer wieder starten, da die *JobQueue* mit dem Schließen des Bots geleert wird.



Wir verwenden eine ähnliche Herangehensweise wie bei der vorhin beschriebenen *stop\_timer –* Methode. Der größte Unterschied besteht in der Verschachtelung zweier *for* – Schleifen, welche nun alle Schulden mit allen Jobs vergleichen und die Kontrollvariable auf *True* setzen, sobald der Job als vorhanden befunden wurde.

Problem bei der Entwicklung

Finden einer geeigneten Vorgehensweise:

Nach etwas Recherche stellte sich die JobQueue als die geeignetste Umsetzungsmöglichkeit heraus, da die Methoden am übersichtlichsten gestaltet waren.

Arbeiten mit dem *context*:

Zu Beginn bat der *context* ein großes Maß an Verwirrung, denn wir fanden anfangs keine klare Definition desselben. Später stellte sich heraus, dass dieser ein beliebiges Objekt beinhalten kann. Dadurch wurde uns der Aufruf des *callbacks* und die Übergabe der Schuld um einiges vereinfacht.