



Jonas
FREIKNECHT

KI-Sprachassistenten mit Python entwickeln

Datenbewusst, Open Source
und modular



Zahlreiche KI-Anwendungsfälle

HANSER

Freiknecht

KI-Sprachassistenten mit Python entwickeln



Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter:

www.hanser-fachbuch.de/newsletter



Jonas Freiknecht

KI-Sprachassistenten mit Python entwickeln

Datenbewusst, Open Source
und modular

HANSER

Der Autor:
Jonas Freiknecht, Landau
www.jofre.de

Alle in diesem Werk enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt geprüft und getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Werk enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Weise aus der Benutzung dieser Informationen – oder Teilen davon – entsteht.

Ebenso wenig übernehmen Autor und Verlag die Gewähr dafür, dass die beschriebenen Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt also auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Werkes, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Einwilligung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung – mit Ausnahme der in den §§ 53, 54 URG genannten Sonderfälle –, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2023 Carl Hanser Verlag GmbH & Co. KG, München, <http://www.hanser-fachbuch.de>

Lektorat: Sylvia Hasselbach

Copy editing: Walter Saumweber, Ratingen

Coverkonzept: Marc Müller-Bremer, München, www.rebranding.de

Covergestaltung: Max Kostopoulos

Titelmotiv: gettyimages.de/CHRISTOPH_BURGSTEDT/SCIENCE_PHOTO_LIBRARY

Satz: Manuela Treindl, Fürth

Druck und Bindung: Hubert & Co. GmbH & Co. KG BuchPartner, Göttingen

Printed in Germany

Print-ISBN: 978-3-446-47231-0

E-Book-ISBN: 978-3-446-47448-2

E-Pub-ISBN: 978-3-446-47659-2

Für Justus und Elisa

Inhalt

1	Der Sprachassistent in unserem täglichen Leben	1
1.1	Warum ein eigener Sprachassistent?	3
1.2	Immer präsent: Das Thema Datenschutz	4
1.3	Für wen ist dieses Buch gedacht?	6
1.4	Aufbau des Buches	7
2	Die Entwicklungsumgebung	9
2.1	Technologieauswahl	9
2.2	Der richtige Editor	11
2.3	Trennen verschiedener Projekte	12
2.4	Versionsverwaltung über Git	15
2.5	Die Komponenten unserer Anwendung	20
2.6	Erstellen eines Klassengrundgerüsts	21
3	Erzeugung künstlicher Sprache	27
3.1	Einsatz traditioneller Frameworks	28
3.2	Text-To-Speech und Multiprocessing	31
3.3	Trainieren einer eigenen TTS-Engine	35
3.3.1	Einführung in Real Time Voice Cloning	36
3.3.2	Exkurs: Sequence-To-Sequence-Modelle und Attention	39
3.3.3	Vorgehensweise und Herausforderungen	44
3.3.4	Einrichten des eigenen TTS-Projekts	46
3.3.5	Beschaffen und Bereitstellen der Trainingsdaten	48
3.3.6	Preprocessing, Training und Evaluation des Encoders	49
3.3.7	Preprocessing, Training und Evaluation des Synthesizers	53
3.3.8	Preprocessing, Training und Evaluation des Vocoders	55
3.3.9	Anwendung der Modelle	56
3.3.10	Fine Tuning des Synthesizers mit der eigenen Stimme	60
3.3.11	Exkurs: Handhabung aller Modulabhängigkeiten	62
3.3.12	Einbinden der Logik in die Text-To-Speech-Klasse	63
4	Spracherkennung	67
4.1	Aktivierungswörter	78

4.1.1	Exkurs: Konfigurationsmanagement	78
4.1.2	Die Hummel und das Stachelschwein	82
4.2	Implementierung einer Spracherkennung	87
4.3	Fingerabdruck der Stimme	89
5	Dialoge und Intentionen	93
5.1	Intent Recognition – Die menschliche Intention verstehen lernen	94
5.1.1	Intent Classifier am Beispiel	95
5.1.2	Intent Parsing am Beispiel	104
5.2	Auswahl eines Chatbot-Frameworks	108
5.2.1	Intents auf Basis regulärer Ausdrücke	112
5.2.2	Intents auf Basis maschinellen Lernens	115
5.3	Modularisierung von Intents	120
5.4	Das Intent Management	126
5.5	Microservice-Organisation von Intents	135
6	Intents entwickeln	141
6.1	Tierstimmen	141
6.2	Wikipedia	146
6.2.1	Question Answering mittels Language Model	149
6.2.2	Exkurs: Transformer-Modelle und Self-Attention	168
6.2.2.1	Der Encoder	169
6.2.2.2	Der Decoder	182
6.3	Erinnerungsfunktion	185
6.3.1	Anpassen der Lautstärke	186
6.3.2	Verarbeiten eines Erinnerungsbefehls	187
6.4	Steuern der Lautstärke	197
6.4.1	Intent zur Lautstärkeregelung	199
6.5	Abspielen von Streams	204
6.5.1	Die Klasse Audioplayer	204
6.5.2	Integration des AudioPlayer	208
6.5.3	Der Intent Webradio	210
6.5.4	Fuzzylogik	214
6.5.5	Die Levenshtein-Distanz	219
6.6	Wetterabfrage	223
6.6.1	Einschränken des Intent-Zugriffs	224
6.6.2	Die Wetterabfrage	228
6.6.3	Time Series Forecasts mit Wetterdaten	232
6.6.3.1	Ausflug in die Welt der Regression	233
6.6.3.2	Beschaffung von Wetterdaten	238
6.6.3.3	Autoregression, Moving Average Model und ARIMA	243
6.6.3.4	LSTMs für Time Series Prediction	256
6.7	Steuerung von Smart-Home-Geräten	268

6.8	Q20-Ratespiel	271
6.9	Passwortverwaltung	280
6.10	Weitere Ideen	286
7	Ein simples User Interface	287
7.1	Einrichten eines Tray-Icons	288
7.2	Anzeigen von Notifications	295
8	Paketieren der Anwendung	299
8.1	Exportieren und Wiederherstellen eines Environments	299
8.2	Erstellen von Binaries	301
8.3	Erstellen eines Installers	309
9	Abschließende Worte	311
9.1	Wohin mit meinen neuen Fähigkeiten?	312
9.2	Danksagungen	315
	Literatur	317
	Stichwortverzeichnis	319

1

Der Sprachassistent in unserem täglichen Leben

Der Sprachassistent – ein meist kleiner, optisch ansprechender Computer – ist mittlerweile fester Bestandteil vieler Haushalte. Der Grund für die rasche Verbreitung ist schnell gefunden: Die Interaktion via Sprache ist um ein Vielfaches leichter, als ein kompliziertes Interface zu bedienen. *Licht Wohnzimmer* spricht sich zügiger und intuitiver aus als in einem User Interface (UI) das Wohnzimmer zu suchen und den Lichtschalter zu drücken. Außerdem muss niemand zu einem Bedienfeld laufen und eine Taste drücken oder das Handy zücken, denn unsere Stimme ist im wahrsten Sinne des Wortes *wireless*.

Auch die Erweiterung eines Sprachassistenten ist denkbar einfach. In der Regel fragt ein Gerät, ob eine bestimmte Funktion aktiviert werden soll, wenn erkannt wird, dass diese eine bestimmte Aufgabe übernehmen kann. An einem PC oder Smartphone müsste man nun ein Programm suchen, das dieser Aufgabe gewachsen ist und es gegebenenfalls installieren und lernen, es zu bedienen. Möchten Sie aber etwa eine Einkaufsliste für die Geburtstagsfeier Ihrer Tochter anlegen und auf Ihr Smartphone synchronisieren, dann machen Sie das in der Regel über einen einfachen Befehl, der automatisch die Funktion *Einkaufsliste* aktiviert.

Doch nicht nur für Benutzer ist der Einsatz eines Sprachassistenten eine Erleichterung. Da Sie sich dieses Buch ansehen, gehe ich davon aus, dass Sie (freiwillig oder unfreiwillig) mit dem Gedanken spielen, ein derartiges Gerät zu entwickeln, und sich auch damit auseinandersetzen müssen, gewisse Funktionen dafür zu programmieren (wir werden das in Kapitel 6 ausgiebig tun). Und auch hier kommt uns die menschliche Sprache gelegen, denn ein umständliches Design eines User Interface entfällt für die Entwickler (es sei denn, Sie entwickeln für den *Echo Show*) und zumindest mir persönlich ist das immer eine der unliebsamsten Aufgaben, muss ein Interface doch auf hunderten verschiedenen Geräten ordentlich aussehen und funktionieren. Also auch als Entwickler profitiert man von der einfachen Interaktion per Stimme. Voraussetzung ist, dass man sich ein Framework schafft, das Stimme versteht, doch auch das werden wir zusammen gemeinsam in diesem Buch in aller Ausführlichkeit tun.

Allerdings hat auch ein Sprachassistent seine Tücken. Allen voran fällt das Thema Datenschutz immer mal wieder unangenehm auf, wenn etwa Transkripte von Interaktionen mit Amazons Alexa auftauchen, die von Dienstleistern abgeschrieben und somit auch gelesen werden. Auch wenn das der Weiterentwicklung und der Verbesserung des Sprachverständnisses dient, bekommt man eventuell ein mulmiges Gefühl im Bauch, da auf den Festplatten eines IT-Giganten gespeichert ist, was im eigenen Wohnzimmer gesprochen wird. Auch diesem Thema widme ich einen Teil dieses Buches, und tatsächlich ist es sogar einer der Gründe, warum ich mich darangesetzt habe, einen eigenen Sprachassistenten zu schreiben. Ich wollte eine Offline-Variante schaffen, die nur für die nötigsten Aufrufe eine Internetverbindung herstellt.



HINWEIS: Es gibt sehr viele Sprachassistenten – Amazon Alexa, Google Home, Mycroft als Open-Source-Variante, Siri von Apple, Samsung Bixby oder Microsoft Cortana. Ich hoffe, Sie sehen es mir nach, dass ich nicht jedes Gerät im Haus aufgestellt habe, um darüber berichten zu können – die fertigen Produkte sollen uns ja nur als Orientierungshilfe dienen. Demnach werden meine Beispiele in der Regel an Amazons Alexa dargestellt, wobei das keinerlei Qualitätsmerkmal dieser einen Ausführung ist. Sie war einfach zuerst da.

Des Weiteren ist der Zugriff auf einen Sprachassistenten in der Regel nicht eingeschränkt. Die Einkaufsfunktion von Alexa beispielsweise ist ab Werk nicht deaktiviert. Das bedeutet, dass jeder, der in Rufweite des Geräts steht, Einkäufe tätigen kann. Diese Erfahrung habe ich bereits am eigenen Leib gemacht, denn bis ich das entsprechende Häkchen in den Optionen der App gefunden hatte, hatten meine Kinder schon zwei Monate lang eine zusätzliche Musikoption gebucht. Eine andere Anekdote hat ein ehemaliger Arbeitskollege von IBM erzählt, dessen Töchter sich wohl seit einiger Zeit die Mathehausaufgaben vom heimischen Assistenten erledigen ließen, um mehr Zeit zum Spielen zu haben. Von den Kollegen, die im Büro die Lautstärke auf 10 stellen und DJ Bobo spielen lassen, fange ich gar nicht erst an.

Es zeichnet sich also ab, dass Kontrollmechanismen, die nicht nur stur Befehle interpretieren und ausführen, einen wirklichen Mehrwert bringen würden. In einigen Fällen kann es gewünscht sein, dass nur eine bestimmte Person eine bestimmte Funktion ausführen darf, etwa das bereits angesprochene Einkaufen oder auch die Administration des Sprachassistenten. Aber auch im Bereich der Hausautomatisierung, in dem der räumliche Aspekt eine Rolle spielt, kann es hilfreich sein, z. B. zwischen Stimmen unterscheiden zu können, sodass nicht durch Fehlinterpretation eines Funktionsaufrufs im Kinderzimmer nachts der Rollladen hochgefahren oder das Licht eingeschaltet wird. Besagte Kontrollmechanismen werden wir in Abschnitt 6.6.1 besprechen und auch praktisch implementieren.

Letzteres ist ein gutes Stichwort, um darauf hinzuweisen, wie wir in diesem Buch arbeiten. Ich möchte mit Ihnen die Themen nicht nur trocken durchsprechen, sondern nach und nach implementieren, um herauszufinden, wie wir bestimmte Funktionen bestmöglich umsetzen können. Besonders im Bereich der künstlichen Intelligenz (KI; ich mag diesen Begriff nicht besonders, werde ihn aber dennoch verwenden) möchte ich durch die praktische Anwendung versuchen, für Klarheit zu sorgen und zu zeigen, was sich hinter dem Begriff *Intelligenz* im Kontext von Sprachassistenten oder Assistenzsystemen im Allgemeinen verbirgt, inwiefern man von einem eigenständig denkenden System sprechen kann und wie nah wir einer allgemeinen KI (AGI; Artificial general intelligence) mit unserem Projekt kommen können.



HINWEIS: Zu diesem Buch gibt es ein *GitHub* Repository mit der Code-Basis für die einzelnen Kapitel, sodass Sie auch ohne Schreiarbeit den Fortschritt jedes Abschnitts nachvollziehen können. Sie finden es hier: <https://github.com/padmalcom/AISpeechAssistant>

Ziel des praktischen Teils soll sein, dass Sie am Ende des Tages einen eigenen Sprachassistenten implementiert haben, den Sie mit Ihrer Stimme steuern und beliebig um Funktionen erweitern können.

■ 1.1 Warum ein eigener Sprachassistent?

Nun könnten Sie natürlich auch einfach in den nächstbesten Elektronikgroßhandel gehen und sich einen der diversen Geräte aus dem Hause Google, Amazon, Samsung, Apple etc. kaufen. Warum also sollten Sie sich selbst die Mühe machen und einen Assistenten entwerfen? Hierfür gibt es einige gute Gründe.

Dieses Buch ist in seiner Gesamtheit ein Werk, das Sie von Beginn an anhand eines praktischen Anwendungsfalls durch die Applikationsentwicklung in Python führt und bis auf wenige Ausnahmen jedes Thema behandelt, mit dem Sie u. a. im Kontext Data Analytics, Natural Language Processing (NLP), Audio Processing, Deep Learning, Machine Learning, Kompilierung, Logging, Error Handling etc. in Berührung kommen. Kurzum: Es bietet Ihnen die Möglichkeit, Ihre Entwicklerfähigkeiten auszubauen und zu festigen.

Natürlich werden wir uns auch, wie bereits gesagt, dem Thema KI widmen und die Unterschiede zwischen regelbasierten Entscheidungen und maschinellem Lernen kennenlernen, ganz konkret, wenn es darum geht die Befehle der Sprecher zu interpretieren. Ebenso schauen wir uns den Bereich Deep Learning im Kontext Sprachsynthese, NLP und Zeitreihenvorhersagen an. Wir werden in Kapitel 6 auf verschiedene Architekturen (*Recurrent Neural Networks* (RNN), *Long Short-Term Memory* (LSTM) und *Transformer*) eingehen und lernen, wie diese funktionieren, wie sie aufeinander aufbauen und wie wir sie für unsere Zwecke nutzbar machen können. Eine Einschränkung muss ich jedoch machen: Den generellen Aufbau von neuronalen Netzen und die diesen zugrunde liegenden Mechanismen, wie etwa Optimierungsfunktionen à la *Gradient Descent*, werden wir nicht durchsprechen, sondern als gegeben hinnehmen. Da gibt es bessere Bücher und Onlinekurse, die sich diesen Themen in der notwendigen Ausführlichkeit widmen. Jedoch werden wir uns anschauen, wie wir Trainingsdaten erzeugen, bereinigen und das Training mehrerer neuronaler Netze durchführen, um zu lernen, wie wir die Güte und den Fortschritt des Trainingsvorgangs beurteilen können und wie wir die fertig trainierten Modelle in unseren Sprachassistenten einbinden.

Ein weiterer Grund, einen eigenen Sprachassistenten zu schreiben, sollte sein, dass Sie die Hoheit über Ihre eigenen Daten haben; Datensouveränität hat sich dafür als Begriff in den letzten Jahren durchgesetzt. Ich möchte hier ausdrücklich betonen, dass wir in diesem Buch keinerlei Cloud-Dienste nutzen, sondern alle Berechnungen *on-premises* durchführen. *On-premises* bedeutet, dass Vorgänge lokal auf dem von Ihnen genutzten Gerät stattfinden und nicht an entfernte Geräte ausgelagert werden. Natürlich gibt es Ausnahmen, z. B. wenn wir eine Funktion schreiben, die das Wetter abfragt, unseren Standort feststellt oder Streams abspielt. Aber dann senden wir nur die nötigsten Informationen. Und Ihr Sprachassistent funktioniert auch dann, wenn keine Internetverbindung besteht.

Bei der Verarbeitung der Audioeingabe bleibt es also Ihnen überlassen, ob Sie Texte mit-schneiden möchten, etwa um die Befehlsverarbeitung zu verbessern. Wir werden das in diesem Buch nicht tun, da wir von sogenannten Aktivierungswörtern (*Wake Words*) Gebrauch machen werden, die die Interpretation der Sprache erst aktivieren, wenn ein bestimmtes Wort gesprochen wird. Weiterhin machen wir den Zustand des Assistenten zu jeder Zeit durch ein Icon sichtbar, sodass Sie und alle, die Ihren Assistenten später verwenden, sicher sein können, wann er wartet, lauscht, denkt oder spricht. Die Offline-Fähigkeit hat noch einen anderen Vorteil: Der Assistent wird portabel. Natürlich muss für die entsprechende Stromversorgung gesorgt werden, jedoch sind wir nicht davon abhängig, dass eine Internetverbindung besteht.

Eine weitere Herausforderung, die auch für mich als Motivation diente, ist, den Assistenten auf deutscher oder besser nicht englischer Sprache zu implementieren. Viele fertige Bibliotheken, die mit Sprache oder Text arbeiten, sind auf das Englische ausgerichtet und funktionieren überragend. Doch wenn es dazu kommt, eine Umsetzung in Spanisch, Französisch oder wie in unserem Falle Deutsch anzugehen, stoßen wir bei vielen fertigen Bausteinen schnell an die Grenzen. Und diese Grenzen gilt es zu evaluieren und zu durchbrechen, entweder mit Workarounds oder mit Fleiß und Schreiarbeit. Sollten Sie nicht hauptberuflich Softwareentwicklung betreiben, lernen Sie hier, mit welchen Problemen man in diesem Berufsfeld häufig konfrontiert wird und wie diese zu lösen sind.

Der letzte Punkt, den ich hier anführen möchte, ist Spaß. Mir hat es Freude gemacht herauszufinden, wie die Entwickler im Silicon Valley bestimmte Probleme gelöst und welche Fehler sie sicher zu Beginn gemacht haben. Auch bei mir war der Lerneffekt hoch, denn meistens ist man ja als Spezialist auf eine Domäne ausgerichtet (Data Science, Web, Backend, Data Engineering ...) und hier schauen wir kontinuierlich über den Tellerrand. Durch die modulare Erweiterbarkeit des Assistenten können Sie in beliebige Gebiete reinschauen, die Sie interessieren und entsprechende Funktionen an das System andocken lassen.

■ 1.2 Immer präsent: Das Thema Datenschutz

Ich habe einen Freund, der Sicherheitsprüfungen für Firmen durchführt und Zertifizierungen ausstellt. Auch wenn es in seiner täglichen Arbeit mehr um Sicherheitsaspekte von Arbeitsausrüstung und Maschinen geht, war seine Reaktion auf die Alexa (richtigerweise *Amazon Echo*; *Alexa* bezeichnet nur den Assistenten, nicht das Device) in unserem Wohnzimmer, freundlich ausgedrückt, eher verhalten. Wieso rufen Sprachassistenten bei vielen Menschen eine derartige Reaktion hervor?

Sucht man im Internet nach *Alexa Voice Transcripts*, wird man schnell fündig und erfährt, was Amazon von den Benutzereingaben für die Verbesserung seiner Dienste verwertet und was nicht. So wurde 2019 bekannt, dass sich zwar über die Konfiguration einstellen lässt, dass eigene Stimmufnahmen gelöscht werden, jedoch die übersetzten Befehle weiter gespeichert werden (Kelly & Statt, 2019). Zwar sind diese häufig harmlos, aber der Teufel steckt bekanntlich im Detail und wenn nur eine von 100.000 Transkripten eine Kontonummer, Geschäftsgeheimnisse oder Krankendaten beinhaltet, ist die Speicherung in Frage zu stellen.

Und haben Sie schon mal erlebt, dass Ihr Assistent angesprungen ist, obwohl Sie das Aktivierungswort nicht gesprochen haben? Der Klassiker bei uns ist die Frage *War lecker?*, die als *Alexa* interpretiert wird (mit Kindern sagt man diesen Satz öfter, als man denkt) und schon beginnt das Gerät aufzuzeichnen und sendet Daten, auch wenn es das in dem Moment nicht soll und die Nutzer auch nicht damit rechnen, dass es das tut.

Ein noch viel einfacherer Fall passiert in unserem Haushalt recht häufig. Bestellt man bei Amazon und erhält in Kürze eine Lieferung, hat Alexa die Angewohnheit, gelb zu leuchten, bis man per *Benachrichtigung* abfragt, welcher Hinweis denn vorliegt. Das Team rund um Amazons Sprachassistenten ist so schlau, dass es diese Funktion über Weihnachten deakti-

viert, jedoch stehen ja die restlichen elf Monate jede Menge Geburtstage und Feierlichkeiten an. Und so ist hier und da schon im Vorfeld bekannt geworden, was eine bestimmte Person geschenkt bekommt. Das Thema mag man jetzt mit einem Schmunzeln abtun, ist es doch kein großes Geheimnis, wo die Geburtstagsgeschenke herkommen. Doch haben Sie mal darüber nachgedacht, dass Ihrem Assistenten gegebenenfalls Ihre Einkaufshistorie bekannt ist? So passiert es seit Kurzem, dass eine Benachrichtigung angezeigt wird, die mich fragt, ob ich einen Artikel bewerten möchte, den ich kürzlich gekauft habe. Hier wird unmissverständlich klar, dass Sprachassistenten auch Daten und somit im entferntesten Sinne Umsatz generieren sollen und nicht nur das tägliche Leben erleichtern.

Die eben angesprochene Tatsache, dass Amazon Audioaufnahmen von uns vorliegen, sagt einiges über den Verarbeitungsprozess der Daten aus. Es hat den Anschein, dass die Logik zur Verarbeitung des gesprochenen Befehls nicht auf der Hardware selbst liegt, sondern in der Cloud (oder für Realisten: in einem der Rechenzentren von Google, Microsoft, Amazon etc., siehe Bild 1.1). Das schicke Gerät, das wir in unseren Räumen stehen haben, ist in der Regel also nur ein Datentransporteur oder Proxy, der zwar die Aktivierungswörter wahrzunehmen versteht, den Mitschnitt aber an einen Cloud-Service streamt, um ihn dort auch zu prozessieren.

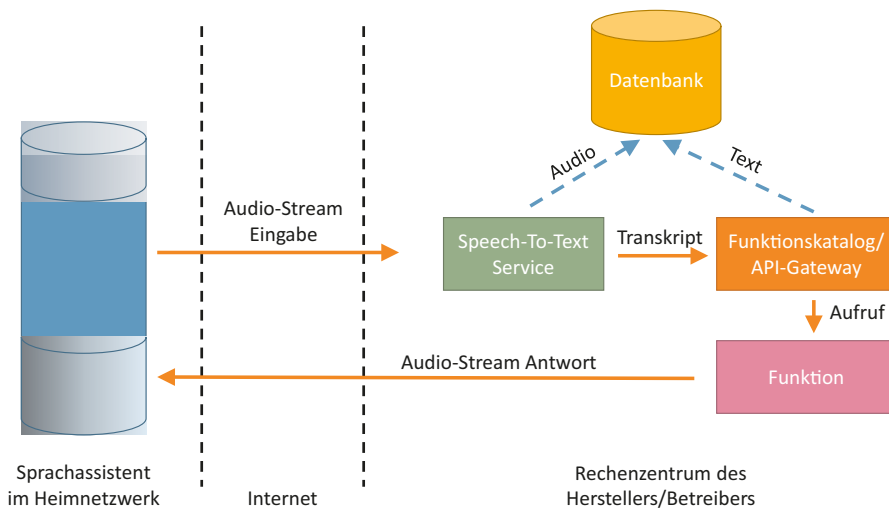


Bild 1.1 Datenfluss bei Aufruf eines Sprachassistenten mit Backend in der Cloud des Herstellers

Die Begründung für dieses Vorgehen liegt aber nicht nur im so oft bekundeten Datenhunger der größeren Anbieter. Wenn Sie den Betrieb eines Geräts, das beim Kunden steht, so minimalistisch wie möglich aufbauen, haben Sie dort auch weniger Aufwand zu leisten. Ganz konkret reduzieren Sie die Anzahl der möglichen Fehlerquellen auf dem Gerät und verlagern diese stattdessen in Ihr eigenes Rechenzentrum, in dem Sie auf alle Daten, Protokolle und Anwendungslogiken Zugriff haben. So ist es viel leichter, Fehler zu beheben und Updates einzuspielen oder experimentelle Features an einer kleinen Benutzergruppe zu testen, als wenn Sie möglicherweise durch ein einzelnes Update eine Reihe Geräte beschädigen, sodass die Benutzer sie erst manuell zurücksetzen müssen.

Nun wechseln wir mal die Seiten. Wir werden schließlich in den nächsten Tagen zu den Entwicklern, die ebenfalls einen Assistenten entwickeln wollen, und kommen somit nicht darum herum, Verständnis für das Speichern von Befehlen aufzubringen. Schließlich möchten wir ja wissen, wie mit unserem Assistenten interagiert wird. Die Herausforderungen dabei sind vielfältig.

Unsere Sprache, Intonation und Sprechgeschwindigkeit ist sehr divers, wir haben alleine in Deutschland schon zwanzig Dialekte und diese haben ein ganz eigenes Vokabular, das selbst ein Mensch erst über Jahre erlernen muss (ich bin als Hannoveraner in die Pfalz gezogen; ich weiß, wovon ich spreche) und selbst wenn es keine Dialekte gäbe, sind die Sprechweisen ganz anders. So muss *Wetter Landau* dasselbe Ergebnis liefern wie *Wie ist das Wetter in Landau?*, *Gummibärchen einkaufen* wie *Erinnere mich daran, Gummibärchen zu kaufen* und *Still* dasselbe wie *Stopp, Ruhe* oder womöglich *Schweigefuchs*. Sie sehen also, dass Daten unerlässlich sind, um das Verständnis der Intentionen der Benutzer zu verbessern und aktuell zu halten. In Kapitel 6, wenn wir kleine Modelle anlernen, die verschiedene Formulierungen von Befehlen verstehen, und Sie grübelnd vor dem PC sitzen und überlegen, wie ein Benutzer eine bestimmte Funktion aufrufen könnte, würden Sie sich eine Tabelle mit den zwanzig häufigsten Formulierungen wünschen, das verspreche ich Ihnen. Und genau das ist ein Verwendungszweck für die gesammelten Daten.

Natürlich ist das kein Freibrief und auch keine gute Begründung dafür, persönliche Gespräche mitzuschreiben, doch vielleicht zumindest eine Erklärung. Historisch ist der Umstand wohl so zu begründen, dass in den USA erst die Innovation kommt, dann die ethischen und datenschutzrechtlichen Bedenken und Regeln. In Deutschland ist es umgekehrt, wir machen erst die Regeln und schränken uns dadurch häufig sehr ein – und erwarten das natürlich auch von den Ländern, die uns ihre Technologie liefern. Ein klassischer *Clash of Cultures*.

■ 1.3 Für wen ist dieses Buch gedacht?

Die Frage ist gar nicht so leicht zu beantworten, denn im Prinzip ist dieses Buch für all diejenigen hilfreich, die einfache Aufgaben in ihrer Umgebung per Stimme automatisieren möchten. Mancher möchte vielleicht durch die Wetteransage geweckt werden, ein anderer möchte sein Garagentor per Sprachbefehl öffnen, eine Dritte möchte das WLAN im Auditorium abschalten, weil ihre Vorlesung beginnt.

Falls Sie die Herausforderung etwas generalistischer betrachten, ist es für Sie vielleicht auch nur interessant, den Entwicklungsprozess eines eigenen Sprachassistenten mit mir durchzugehen, die Funktionsweise zu verstehen und zu lernen, wie sich etwas Ähnliches in Python umsetzen lässt.

Vielleicht kommen Sie aber auch aus einem kleinen Start-Up und haben eine absolut einzigartige Geschäftsidee, die es erfordert, dass Sie ein kleines IoT-Device (Internet of Things) per Sprache steuern können.

Schließlich hoffe ich, dass dieses Buch auch von dem ein oder anderen zur Unterhaltung gelesen wird, denn theoretische Bücher gibt es genug und zumindest mich inspirieren tech-

nisch praktische Werke oft mehr, als solche, die nur Luftschlösser bauen. Mein Bestreben ist, besonders die herausfordernden Themen, dazu zählen ich vor allem die KI-Architekturen, so zu erklären, dass sie verständlich sind und – viel wichtiger – klar wird, wozu diese dienen und wie sie uns in unserem ganz konkreten Fall weiterhelfen. In dem Kontext sollen auch die Neuerungen des maschinellen Lernens der letzten Jahre angesprochen werden, weswegen wir u. a. auch das Thema Quantum Machine Learning kurz betrachten und ausprobieren – es aber nicht in unseren Assistenten einfließen lassen.

Wichtig ist, dass Sie einige Dinge mitbringen. Sie sollten bereits ein wenig Programmiererfahrung haben, denn ich werde nicht bei Variablen und Schleifen beginnen, sondern die Seiten nutzen, um tiefer einzusteigen. Ich werde für alle Beispiele einen Windows-PC verwenden, jedoch ist Python dankenswerterweise in hohem Maße betriebssystemunabhängig, sodass der Assistent auch auf macOS, Ubuntu oder CentOS laufen sollte. Sollten Sie planen, das Training der TTS-Engine selber durchzuführen, benötigen Sie weiterhin eine NVIDIA-Grafikkarte mit mindestens 8 GB Speicher. Alternativ können Sie für das Training auch einen Cloud-Dienst nutzen, etwa *Google Colab*. Dieser stellt GPUs (Graphics Processing Units) oder TPUs (Tensor Processing Units) für das Training neuronaler Netze kostenlos zu Verfügung. Da dieser Teil jedoch optional ist, kann er auch getrost übersprungen und auf andere Frameworks oder meine vortrainierten Modelle zurückgegriffen werden.

■ 1.4 Aufbau des Buches

Bisher habe ich meine Bücher immer so aufgebaut, dass ich die darin besprochenen Themen aufbereitet und sequenziell besprochen habe. Das hatte den Vorteil, dass ich pro Thema beliebig in die Tiefe gehen konnte, ohne den Faden zu verlieren – es gab ja auch keinen. Diesmal möchte ich es etwas anders machen und mich mit Ihnen dem dann doch sehr technischen Inhalt von der fachlichen Seite nähern, sprich wir entwickeln einen Sprachassistenten und Sie lernen dabei entweder in den Hauptkapiteln oder in ergänzenden Abschnitten vertieft die Techniken kennen, die einfach anmutenden Bibliotheken zugrunde liegen. Damit haben wir ein klares Ziel vor Augen, nämlich den Assistenten fertigzustellen, und lernen aber gleichzeitig noch einiges über neuronale Netze, die maschinelle Verarbeitung von Sprache, Trainingsdaten für KI-Modelle usw.

Besagter roter Faden gestaltet sich nun so: In Kapitel 2 lernen Sie, wie man eine solide Entwicklungsumgebung in Python aufsetzt und in einer kurzen Auffrischung das Grundgerüst unserer Applikation erstellt. In Kapitel 3 widmen wir uns der Sprachsynthese, also der Wiedergabe von Sprache über den Computer. Kapitel 4 behandelt die Spracherkennung sowie die Verwendung von Aktivierungswörtern. Ebenso werden Sie verstehen lernen, wie der Fingerabdruck einer Stimme zu erkennen und für die Benutzerauthentifizierung verwendet werden kann. Darauf folgt die Interpretation von Text und das Erkennen der Intention der Sprecher in Kapitel 5. Wir werden hier weiterhin die Unterschiede zwischen regelbasierten und Machine-Learning-Ansätzen ermitteln und schauen, wann welches Verfahren zum Einsatz kommen sollte. In Kapitel 6 beginnt das Handwerk. Wir haben bereits ein robustes Framework implementiert und entwickeln nun dafür die entsprechenden Funktionen.

Da Funktion ein fester Begriff aus der Programmierung ist und ich Missverständnisse vermeiden möchte, nennen wir diese ab hier *Intents*. In einigen dieser Intents werden wir das Framework unseres Sprachassistenten erweitern, etwa um eine Lautstärkeregelung, die Verwendung von Callbacks in Intents, Dialoge über mehrere Fragen und Antworten hinweg und vieles mehr. In Kapitel 7 implementieren wir eine minimale UI, um von der Konsolenanwendung wegzukommen und Sie lernen, wie Toasts und Benachrichtigungen in Windows zu verwenden. Abschließend möchte ich in Kapitel 8 zeigen, wie eine Python-Anwendung kompiliert und paketiert werden kann, sodass Sie mit einem fertigen Installer zu Ihren Freunden und Kollegen gehen und Ihren eigenen Assistenten präsentieren können. Kapitel 9 beinhaltet dann die obligatorische, tränenreiche Verabschiedung und der Ausblick auf die Perspektiven, Studien- und Berufsfelder, denen Sie sich nach dieser Lektüre widmen können.

2

Die Entwicklungs- umgebung

Eine Entwicklungsumgebung enthält prinzipiell erst einmal eine IDE (Integrated Developer Environment), die den Entwicklungsprozess durch einen Editor samt Syntaxhervorhebung, Build- und Versionsverwaltungswerkzeuge sowie die Codeanalyse unterstützt. Heutzutage verfügt jede gute IDE weiterhin über Werkzeuge zur Codeoptimierung und -Vervollständigung, sodass den Entwicklern möglichst viel Arbeit abgenommen wird. Dabei werden Typüberprüfungen automatisch durchgeführt, veraltete Pakete gesucht oder potenzielle Ausnahmefehler (*Exceptions*) aufgezeigt. Kurzum, dem Entwickler wird jegliche Arbeit abgenommen, sodass er sich bestmöglich auf die Anwendungslogik konzentrieren kann.

Zusätzlich machen uns in einigen Programmiersprachen Umgebungs- und Paketmanager das Leben leichter. Sie helfen uns, mehrere Projekte logisch voneinander zu trennen und bieten Funktionen an, um Abhängigkeiten per Mausklick oder Befehl für eine ganz bestimmte Laufzeitumgebung zu installieren, sodass wir auch hier nicht selber mühselig auf die Suche nach den richtigen Paketen gehen und diese herunterladen müssen.

Um genau diese beiden Bestandteile wollen wir uns in diesem Kapitel kümmern. Ist das geschehen, schauen wir uns an, wie ein Environment in Python aufgesetzt wird und wie wir die Anwendungsarchitektur des Sprachassistenten gestalten wollen. Zu guter Letzt wiederholen wir kurz ein paar Basics, um ein einfaches Grundgerüst für den Assistenten aufbauen zu können.

■ 2.1 Technologieauswahl

Lassen Sie uns kurz in die Abgründe der Glaubenskriege um die beste Programmiersprache hinabsteigen und ganz rational überlegen, warum wir für unser Projekt Python einsetzen. Schließlich hätten wir auch C#, Go, Kotlin oder Java verwenden können, oder? Wenn man bereit ist, etwas Mehraufwand in Kauf zu nehmen oder die Architektur des Sprachassistenten z. B. auf einen Microservice-basierten Ansatz (siehe Abschnitt 5.5) umstellt, ja. Da wir aber alle analytischen Funktionen in einer einzelnen Anwendung kapseln wollen, führt kaum ein Weg an Python vorbei. Die Sprache hat sich einfach zu sehr im analytischen Bereich etabliert, als dass man sie durch eine beliebige andere ersetzen könnte (ich selber komme aus der Java-Welt und musste mich auch erst mal umstellen). Es geht hier weniger um die Syntax. Vielmehr sind es die Pakete, die andere Entwickler bereitstellen und die wir für die

Vielzahl von Aufgaben heranziehen, die wir während der Umsetzung bewältigen müssen. Eine Umfrage von Kaggle aus dem Jahre 2020 zeigt deutlich, dass Python mit fast 35 % am stärksten vertreten ist (siehe Bild 2.1).

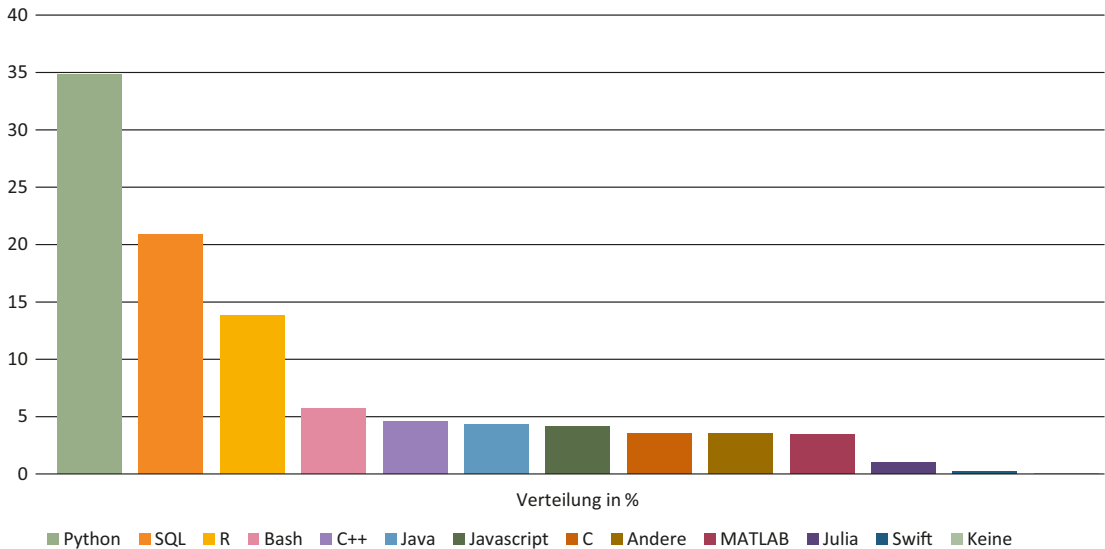


Bild 2.1 Von Data Scientists verwendete Programmiersprachen (Kaggle, 2021)

Sie werden sehen, dass es mit der Entscheidung für Python aber noch nicht getan ist. Im Machine Learning und besonders im Deep Learning entbrennt der nächste Konflikt. Welche Frameworks sollen verwendet werden, Tensorflow, PyTorch, JAX oder fast.ai? Ich habe mich entschieden, mich hier nicht festzulegen, denn häufig existieren vorgefertigte Modelle und Architekturen für einen bestimmten Anwendungsfall, die diktieren, welches Framework zu verwenden ist. Und das ist okay, geht es uns doch um die Umsetzung und nicht um eine Forschungsarbeit.

Sollten Sie sich in der Situation befinden, dass Sie in Ihrem Unternehmen für die Technologieauswahl für KI-Projekte verantwortlich sind, möchte ich Ihnen abermals den jährlichen Kaggle Survey ans Herz legen. Kaggle, als eine Plattform für Data Scientists unter der Schirmherrschaft von Google, fragt dort regelmäßig ab, wer sich zu einem KI-Spezialisten entwickelt, wie die Altersverteilung ist, wo man sich entsprechend weiterbilden kann und eben auch, welche Tools zum Einsatz kommen. Dort ist wunderbar zu erkennen, dass es keine teuren Werkzeuge von IBM und co. braucht, sondern dass weit über die Hälfte der Teilnehmer mit ganz normalen Open-Source-Frameworks arbeiten. Dem zum Trotz arbeiten jedoch wiederum etwa 80 % auf einer Cloud-Plattform der drei Anbieter Amazon, Google und Azure. Es zeigt sich also, dass Machine-Learning-Projekte nicht mit Zaubertools umzusetzen sind, sondern durch die Arbeit von speziell ausgebildetem Personal. Behalten Sie das bitte im Hinterkopf, sollte Ihre Chefin mal danach fragen.

■ 2.2 Der richtige Editor

Ich habe mal an einem Austausch mit der Anglia Ruskin University in Cambridge teilgenommen, was mir einerseits gezeigt hat, dass Cambridge eine wunderschöne, interessante Stadt ist und andererseits, wie unterschiedlich Programmiervorlesungen aussehen können. Von meiner Hochschule in Mannheim war ich es gewohnt, Programmieraufgaben mit nach Hause zu bekommen und in Ruhe lösen zu können. Dort war es etwas anders. Wir wurden in Zweiergruppen vor ein leeres Notepad gesetzt, das weder über Codeanalysefunktionen noch über Autovervollständigung verfügte, und sollten eine Anwendung in Java schreiben. Der Dozent ging herum und gab Hinweise, wenn wir nicht weiterkamen. Ich war aufgeschmissen.

Wenn ich aber heute eine Java-Vorlesung halten würde, würde ich es genauso machen, denn so lernt man selbstständig zu programmieren. Natürlich braucht man für gewisse Arbeiten einen Debugger, der es erlaubt, Anwendungen zu unterbrechen und den Zustand von Variablen zu prüfen, doch für ein *Hello World*, einen Taschenrechner oder einen Kalender reicht tatsächlich ein einfacher Editor und der entsprechende Compiler.

So schwer soll es Ihnen aber nicht gemacht werden. Die Editoren für Python bieten mittlerweile ein Set all der Tools, die uns damals verwehrt blieben. Zu den am häufigsten verwendeten gehören sicher Microsofts *Visual Studio Code (VS Code)*, *PyCharm* oder *Spyder* (wobei ich in meiner Abteilung feststelle, dass die Tendenz ganz eindeutig zu *VS Code* geht). Suchen Sie sich gerne einen aus. Ich werde jedoch in *Notepad++* arbeiten und immer ein *Command Line*-Fenster offen haben, da diese Fenster genau die Informationen und Befehle zeigen, die Sie zum Weiterkommen benötigen.

Die Features, auf die Sie bei der Auswahl achten sollten, liste ich im Folgenden auf. Zwar sind nicht alle wirklich notwendig, sie erleichtern aber die Arbeit ungemein.

- Ein Codeeditor mit Syntaxhervorhebung und Autovervollständigung
- Projektorganisation, um alle dem Projekt zugehörigen Dateien in der IDE zu verwalten
- Refactoring Tools, um etwa Variablen- oder Klassennamen und -Eigenschaften projektweit zu ändern
- Ein Debugger, der in der Lage ist, die Anwendung an bestimmten Punkten zu unterbrechen und den Einblick in Variableninhalte gewährt
- Eine Test-Suite, die das Schreiben und Ausführen von Tests ermöglicht
- Datenbanktools, um im Editor den Inhalt bestimmter applikationsabhängiger Datenbanken zu prüfen und zu ändern
- Visualisierungswerkzeuge, um den Inhalt bestimmter Variablen (Listen, Matrizen) grafisch darzustellen

In der Regel ist die Empfehlung jedoch, dort zu arbeiten, wo Sie sich am wohlsten fühlen. Wenn Sie keine Anhaltspunkte haben, für welchen Editor Sie sich entscheiden sollten, würde ich Stand 2022 *VS Code* empfehlen.



HINWEIS: Das *Jupyter Notebook* sticht in mehrerlei Hinsicht aus dem Pulk an IDEs heraus, denn es ist einerseits die laut *Kaggle* am häufigsten verwendete Umgebung für Datenexploration und generell Data-Science-Projekte und andererseits ist die IDE webbasiert, sprich Sie arbeiten in Ihrem Browser damit. Das hat den Vorteil, dass sich besagte Notebooks leicht ins Internet hochladen und mit anderen teilen lassen.

Ein Jupyter Notebook besteht aus einzelnen Ausführungsschritten, die einen fertigen Text und optional Grafiken rendern können, sodass jeder Schritt in einer Art Markup präsentiert werden kann (siehe Bild 2.2). Dadurch sind Jupyter Notebooks hervorragend geeignet, um zu unterrichten und Wissen weiterzugeben.

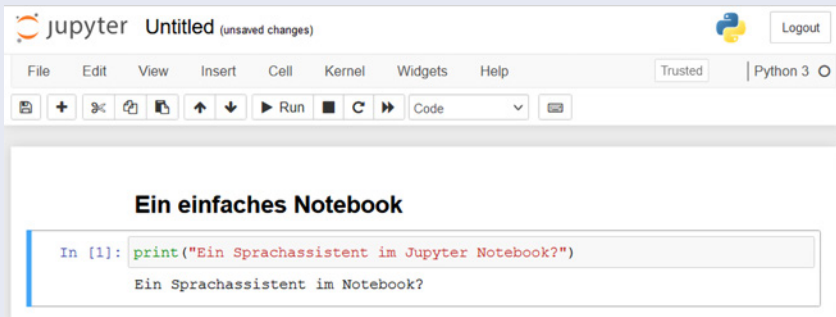


Bild 2.2 Jupyter Notebook eignet sich sehr gut für Datenexploration und Wissensvermittlung.

Da wir aber eher Anwendungsentwicklung betreiben, lohnt sich der Einsatz von Jupyter Notebook maximal für experimentelle Zwecke, also z. B. um zu testen, wie bestimmte Bibliotheken performen oder ob ein komplexes Parsing von Intents das Ergebnis liefert, das wir erwarten. Da wir aber ansonsten das Ziel haben, eine Anwendung zu erstellen, die in Form eines *Executable* selbstständig und mit möglichst geringem Fußabdruck auf den PCs unserer Kunden oder Freunde läuft, werden wir den Großteil der Arbeit in einem konventionellen Editor erledigen.

■ 2.3 Trennen verschiedener Projekte

Ein Python-Projekt besteht in der Regel aus mindestens einer Quelltextdatei, einer Laufzeitumgebung und meistens auch mehreren Bibliotheken, die wir hinzuziehen, da diese Funktionen beinhalten, die wir sonst mühselig selber schreiben und testen müssten. Diese Bibliotheken installieren wir nicht etwa in einen Projektordner, sodass sie nur für ein einziges Projekt vorliegen, sondern eben in diese Laufzeitumgebung. Nun besteht das Problem, dass PCs in der Regel mit einer Python-Version ausgestattet werden und wenn wir nun mehrere Projekte

parallel entwickeln (was recht häufig der Fall sein kann), dann kann es dazu kommen, dass es Überschneidungen zwischen den verschiedenen Bibliotheken und deren Abhängigkeiten gibt (siehe Bild 2.3 links).

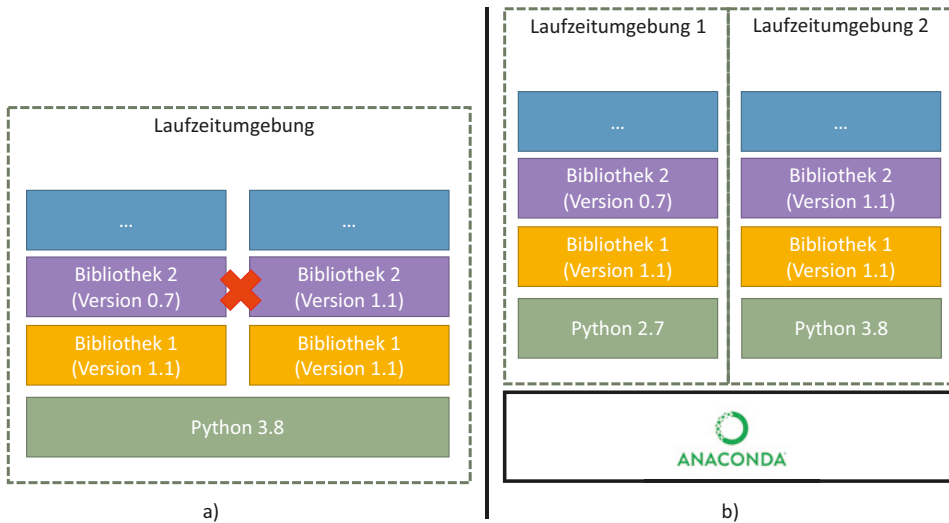


Bild 2.3 Die Trennung verschiedener Umgebungen ist bei der Parallelentwicklung mehrerer Projekte unerlässlich, da es sonst zu Interferenzen zwischen den Abhängigkeiten kommen kann.

Das kann sich dann dadurch äußern, dass beispielsweise Bibliothek X von NumPy 1.21.1 und Bibliothek Y von NumPy 1.19.5 abhängt, und schon würde bei jeder Installation von NumPy die vorherige Installation überschrieben werden und die Funktionalität unserer Anwendungen gefährdet sein.

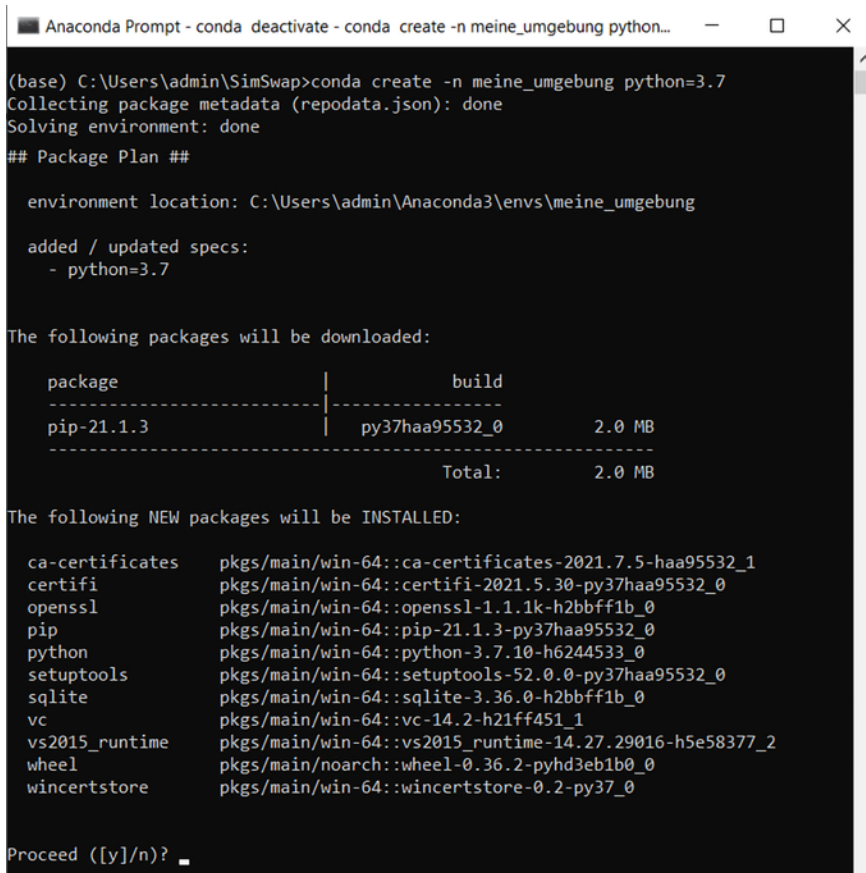
Nun wäre Python nicht so beliebt, wenn es dafür nicht eine einfache Lösungsmöglichkeit gäbe. Diese besteht darin, eine virtuelle Umgebung anzulegen, was wiederum auf mehreren Wegen geschehen kann. *Virtualenv* war lange Jahre der Standard, der über den Befehl `virtualenv meine_umgebung` die Anlage eines eigenen, abgeschotteten Bereichs in der lokal installierten Laufzeitumgebung erlaubte. Über `meine_umgebung\Scripts\activate` konnte diese aktiviert und verwendet werden, sodass alle Bibliotheken, die von nun an installiert wurden, in dieser virtuellen Umgebung landeten.¹

Das Problem, das hierbei bestand, war, dass lediglich eine einzige Version der Laufzeitumgebung unterstützt wurde. Wollte man beispielsweise für Python 2.7 und Python 3.8 entwickeln, musste man beide Umgebungen installieren und dann jeweils auf einer der beiden seine virtuelle Umgebung anlegen. Neben *virtualenv* entstand das Projekt *pipenv*, das einige neue Features mit sich brachte, etwa die Unterscheidung verschiedener Stadien eines Projekts (in der Regel Entwicklung, Test, Integration, Produktion), jedoch nicht das grundsätzliche Problem der verschiedenen Python-Versionen löste.

Irgendwann trat *Anaconda* auf den Plan. Anaconda erlaubt es genauso wie *virtualenv*, virtuelle Umgebungen anzulegen, jedoch mit einigen Besonderheiten. Die Anlage einer neuen

¹ Das gilt nur für Windows-Systeme, auf Linux-Systemen würde man `source meine_umgebung\Scripts\activate` verwenden.

Umgebung über `conda create -n meine_umgebung` erlaubt über einen weiteren Parameter die Spezifikation der Python-Version, die man für genau dieses Projekt nutzen möchte, beispielsweise `conda create -n meine_umgebung python=3.7`. Es werden also beim Einrichten nicht nur die benötigten Abhängigkeiten installiert, sondern auch die Runtime, die wir als Entwickler benötigen. Das Ergebnis sehen Sie in Bild 2.4.



```

Anaconda Prompt - conda deactivate - conda create -n meine_umgebung python...
(base) C:\Users\admin\SimSwap>conda create -n meine_umgebung python=3.7
Collecting package metadata (repodata.json): done
Solving environment: done
## Package Plan ##

  environment location: C:\Users\admin\Anaconda3\envs\meine_umgebung

  added / updated specs:
    - python=3.7

The following packages will be downloaded:

package | build
-----|-----
pip-21.1.3 | py37haa95532_0 2.0 MB
-----|-----
Total: 2.0 MB

The following NEW packages will be INSTALLED:

ca-certificates pkgs/main/win-64::ca-certificates-2021.7.5-haa95532_1
certifi pkgs/main/win-64::certifi-2021.5.30-py37haa95532_0
openssl pkgs/main/win-64::openssl-1.1.1k-h2bbff1b_0
pip pkgs/main/win-64::pip-21.1.3-py37haa95532_0
python pkgs/main/win-64::python-3.7.10-h6244533_0
setuptools pkgs/main/win-64::setuptools-52.0.0-py37haa95532_0
sqlite pkgs/main/win-64::sqlite-3.36.0-h2bbff1b_0
vc pkgs/main/win-64::vc-14.2-h21ff451_1
vs2015_runtime pkgs/main/win-64::vs2015_runtime-14.27.29016-h5e58377_2
wheel pkgs/main/noarch::wheel-0.36.2-pyhd3eb1b0_0
wincertstore pkgs/main/win-64::wincertstore-0.2-py37_0

Proceed ([y]/n)? _

```

Bild 2.4 Anlage eines neuen „Virtual Environments“ über Anaconda und Abfrage der Installation der neu hinzuzufügenden Pakete

Anaconda listet dabei explizit auf, welche Bibliotheken es in der Umgebung installiert. Diese sind oft betriebssystemabhängig und sorgen erst einmal für die Funktionsfähigkeit der Umgebung. So beinhaltet *ca-certificates* etwa notwendige SSL-Zertifikate, um auf verschiedene Datenquellen zuzugreifen, *pip* dient der Installation von Paketen oder *wheel* erlaubt das Arbeiten mit sogenannten Wheel-Dateien (Installationspaketen für Python-Bibliotheken). Bestätigen Sie die Frage aus Bild 2.4, wird Anaconda die benötigten Pakete selbstständig herunterladen oder aus dem lokalen Cache installieren, falls diese schon in aktueller Version vorliegen. Dabei wird die Python-Version in Betracht gezogen, die Sie gewählt haben, denn viele Bibliotheken werden für eine ganz bestimmte Version zusammengestellt.

Danach können Sie die Umgebung per Befehl, den Ihnen Anaconda ausgibt, aktivieren. Dazu kommen wir aber später noch. Sie sehen, dass uns hier die Arbeit abgenommen wird, die uns davon abhalten würde, sofort mit der Entwicklung zu starten.

Nun kommt noch ein Clou von Anaconda: Wir können nicht nur Python-Pakete installieren, sondern auch davon losgelöst andere Binaries, DLLs etc. Später werden wir sehen, wann genau das sinnvoll ist und wie uns damit eine Menge Arbeit abgenommen wird, wenn wir *Ffmpeg* für unseren Streaming-Intent einrichten müssen.



TIPP: Sollten Sie in einer Umgebung arbeiten, die Sie durch verfügbaren Speicherplatz einschränkt, und nur die Grundfunktionalität von Anaconda benötigen, können Sie auf *Miniconda* ausweichen. Miniconda ist Anaconda sehr ähnlich, bringt aber nur die nötigsten Abhängigkeiten mit der Installation mit, sodass der Installer nur etwa 60 MB groß ist, während Anaconda auf etwa 480 MB kommt. Der häufigste Use Case für Miniconda ist das Einrichten von Umgebungen auf einem *Raspberry Pi*.

■ 2.4 Versionsverwaltung über Git

Allem voran: Das ist kein Buch über *Git*. Da ich jedoch den Quelltext dieses Buches auf *Github* hoste, möchte ich Ihnen eine schnelle Einführung in die wichtigsten Befehle geben. *Git* als Versionskontrollsystem hat in den letzten Jahren so sehr an Popularität gewonnen, dass es fast für alle großen Projekte verwendet wird und *SVN* (Subversion) beinahe komplett vom Markt verdrängt hat.



HINWEIS: Github wurde 2018 für 7,5 Milliarden Dollar von Microsoft gekauft und nicht nur weiter kostenlos betrieben, sondern auch um viele Funktionen erweitert. Dennoch gibt es Projekte, die sich von Github fernhalten. Das vielleicht bekannteste ist Blender, ein Werkzeug zur Erstellung von 3D-Modellen, Filmen und Spielen. Die Begründung von Ton Roosendaal, dem Vorsitzenden der Blender Foundation, ist, dass so viel Risikokapital (ca. 350 Millionen Dollar) in das Produkt geflossen ist, dass die Investoren irgendwann Profit zurückbekommen wollen, und er geht davon aus, dass die Benutzer diejenigen sind, die diese Beträge zahlen werden (Roosendaal, 2018).

Also, was ist *Git*? Tatsächlich handelt es sich nur um eine Sammlung von Tools für die Verfolgung von Änderungen an Dateien, sei es Text, Quellcode oder auch Binärdateien. Dokumente lassen sich vergleichen, zusammenführen, aus einem und in ein Repository laden. Ein Repository ist ein Ordner auf Ihrer Festplatte, der alle Daten eines Projekts beinhaltet, die Sie nachverfolgen möchten. Dabei arbeitet *Git* dezentral, sprich es wird kein Server benötigt, der die primäre, einzig wahre Version eines Projekts vorhält. Natürlich macht es aber

Sinn, das zu tun, besonders wenn man, wie in meinem Fall, ein Projekt mit vielen Menschen teilen möchte.

Sollten Sie Git noch nicht auf Ihrem PC installiert haben, laden Sie es von <https://gitforwindows.org> herunter und installieren Sie es. Auf Ubuntu erfolgt die Installation über `apt install git`, auf CentOS analog mit `yum install git`.

Ein neues Projekt anlegen oder ein bestehendes klonen

Wenn Sie ein neues Projekt beginnen möchten, haben Sie im Prinzip zwei Möglichkeiten:

- Sie legen ein neues Projekt mit `git init` an,
- oder Sie klonen ein bestehendes Repository mit `git clone`.

Da ich den Code für den Sprachassistenten bereitstelle, beginnen wir mit Letzterem. Öffnen Sie eine Kommandozeile (*Windows Button* und *cmd* tippen, alternativ **STRG+R** und *cmd* eingeben) und führen Sie folgenden Befehl aus.

```
git clone https://github.com/padmalmcom/AISpeechAssistant
```



TIPP: Besser als die Kommandozeile ist die *Git Bash*, die Sie ebenfalls über das Windows-Startmenü finden oder im Windows-Explorer per Rechtsklick in einen Ordner aufrufen können.

In dem Ordner, in dem Sie sich in der Kommandozeile befunden haben, wird nun ein Ordner *AISpeechAssistant* angelegt, in den *Git* alle Dateien aus eben diesem Repository herunterlädt. Wenn Sie wissen möchten, welche das sind, öffnen Sie den Link hinter `git clone` einfach in Ihrem Browser.

Auch wenn wir `git init` nicht benötigen, möchte ich den Befehl der Vollständigkeit halber noch erklären. Möchten Sie ein neues Repository für ein ganz neues Projekt anlegen (vielleicht möchten Sie ja ohne die Hilfe meines Repositorys arbeiten und den Assistenten von Grund auf neu schreiben, was ich sehr begrüßen würde), dann erstellen Sie einen neuen Ordner über `mkdir neues_repository`, gehen mit `cd neues_repository` in diesen Ordner und führen dort `git init` aus.

Nun können Sie in diesem Ordner neue Dateien, z. B. eine *main.py*, anlegen, die sie später dem Repository hinzufügen.

Sollten Sie ein Projekt mit `git init` angelegt haben, ist es empfehlenswert, dieses mit einem Remote Repository zu verbinden, um mit Ihrem Team gemeinsam daran arbeiten oder von mehreren PCs darauf zugreifen zu können. Dazu müssen Sie zuerst auf einem Git-Server Ihrer Wahl (*GitHub*, *Gitlab*) ein Remote Repository anlegen, indem Sie sich dort einen Account erstellen und den Anlageprozess im Browser durchlaufen. Nehmen wir an, Sie hätten das als Benutzer *ki-entwickler* auf *GitHub* getan und ein Repository namens *sprachassistent* angelegt, dann würden Sie nun in der Kommandozeile folgenden Befehl absetzen, um die Remoteverbindung zwischen Ihrem lokalen Repository und dem auf *GitHub* herzustellen.

```
git remote add origin https://github.com/ki-entwickler/sprachassistent.git
```

Es hat sich eingebürgert, dass das primäre Remote Repository den Namen *origin* bekommt, Sie können hier aber auch frei wählen.



HINWEIS: Wenn Sie ein `git clone` ausgeführt haben, wird das Repository automatisch gesetzt.

Prüfen des Repository-Status

Führen Sie in ihrem Repository nun `git status` aus, gibt der Befehl zurück, ob es Dateien gibt, die nicht nachverfolgt werden oder geändert wurden. Haben Sie eben im Zuge von `git init` die *main.py* angelegt, werden Sie nun darauf hingewiesen, dass diese noch nicht versioniert wurde.

Hinzufügen neuer Dateien in ein Repository

Wenn Sie einem Repository neue Dateien hinzufügen möchten, dann können Sie das über `git add` tun. Indem Sie beispielsweise `git add main.py` ausführen, wird *main.py* dem Repository hinzugefügt und ab sofort von der Nachverfolgung mit einbezogen. Wenn Sie viele Dateien auf einmal hinzufügen möchten, verwenden Sie `git add -a`, wobei *-a* für *all* steht.

Erstellen eines Commits

Ein Commit ist eine Art Repository-Schnappschuss zu einer ganz bestimmten Zeit, etwa wenn wir unsere Datei *main.py* hinzugefügt haben. Achten Sie darauf, dass ein Commit nur möglich ist, wenn es Änderungen an Dateien am Projekt gegeben hat (Änderungen an Ordnern sind für *Git* uninteressant, solange diese keine Dateien beinhalten). Einen Commit erstellen Sie mit `git commit`. Dieser Aufruf wird einen Editor (in der Regel in der Kommandozeile) öffnen, in dem Sie einen Kommentar zu Ihrem Commit eintragen müssen, um die Änderungen zu beschreiben. Alternativ können Sie die Nachricht auch direkt im Befehl über `git commit -m „main.py angelegt“` angeben.

Änderungen hochladen

Haben Sie nun lokal einige Änderungen (also einen oder mehrere Commits) gemacht, können Sie diese über `git push` in das Remote Repository hochladen. In der Regel reichen genau diese zwei Wörter, Sie haben allerdings noch Kontrolle über den Namen des Remote Repositorys und den Branch, auf den Sie pushen möchten (zu *Branches* kommen wir gleich). Voll ausgeschrieben könnte der Befehl `git push origin main` lauten, wenn wir das Remote Repository *origin* auswählen und auf den Branch *main* pushen.



HINWEIS: Github hat den Branch *master* kürzlich aus Political-Correctness-Gründen in *main* umbenannt, was längst überfällig war, da Master/Slave-Beziehungen in anderen Feldern der IT aus dem Sprachgebrauch verbannt wurden.

Ausschluss von Dateien und Ordnern über .gitignore

Nicht immer ist es sinnvoll, alle Dateien in ein Repository hochzuladen. Besonders Binaries und temporäre Dateien sollten auf dem Rechner des jeweiligen Entwicklers verbleiben und erst bei Bedarf wieder generiert werden. Um bestimmte Dateitypen und Ordnerstrukturen von der Nachverfolgung auszuschließen, können Sie eine Datei anlegen, die *.gitignore* heißt und im Root-Verzeichnis des Repositorys liegt.

In einem Python-Projekt sollten beispielsweise folgende Dateien aus einem Repository ausgeschlossen werden:

- `__pycache__` - zu Bytecode kompilierter Python-Code im Ordner `__pycache__`
- `log.txt` - Log-Dateien
- `.spyderproject` - Projektdateien für den Editor Spyder

Die Liste ist natürlich nicht vollständig, sondern nur ein Tropfen auf den heißen Stein. Schauen Sie in das Beispielprojekt, dort finden Sie eine recht umfassende Liste, die teilweise von *Github* generiert werden kann, wenn man dort ein Repository anlegt. Ich habe selber noch einige Einträge am Ende hinzugefügt, die Sie später kennenlernen werden.

Änderungen herunterladen

Es kann vorkommen, dass ich ab und an noch Fehler im Code beseitige und Änderungen in das offizielle Repository pushe. Führen Sie also gerne regelmäßig `git pull` oder `git pull origin main` aus, um den neusten Stand des Codes vom Remote Repository zu bekommen.

Einen Branch erstellen

Nun kann es vorkommen, dass Sie mitten in Ihrem Entwicklungsprozess einen komplett neuen Weg einschlagen möchten, beispielsweise weil Sie eine tolle neue Bibliothek gefunden haben, die Ihnen das Verständnis der Benutzerwünsche erheblich vereinfacht. Diese neue Bibliothek einzubinden bringt aber nun viele Veränderungen im gesamten Programm mit sich, sodass Sie sich Sorgen machen, ob Sie alles im Falle eines Scheiterns wieder zurückgedreht bekommen.

Hier kommen Branches ins Spiel. Aus dem *main*-Branch kann ein `git branch intent_parsing` ausgeführt werden, um den Branch *intent_parsing* zu erstellen. Dieser ähnelt in seinem Zustand dem Branch *main*, kann aber komplett losgelöst davon bearbeitet werden.

In einen Branch wechseln

Möchten Sie nun in den Branch *intent_parsing* wechseln, führen Sie ein `git checkout intent_parsing` aus. Sollten Sie die *Git Bash* verwenden, werden Sie sehen, dass sich der Name des aktiven Branches von *main* auf *intent_parsing* ändert.

Zwei Branches zusammenführen

Prima, Sie hatten Erfolg bei der Implementierung der neuen Funktion und möchten nun das Ergebnis aus dem Branch *intent_parsing* in den Branch *main* übertragen. Hier kommt das Feature *merge* ins Spiel. Commiten Sie Ihre Arbeit im Branch *intent_parsing* und wechseln Sie zurück in den Branch *main* mit `git checkout main`. Sie werden feststellen, dass Sie

ihre Änderungen aus *intent_parsing* nicht mehr sehen. Diese sind aber noch da, nur eben nicht in *main*. Führen Sie ein `git merge intent_parsing` aus. Alle Neuerungen werden nun in *main* übernommen.



HINWEIS: Leider geht ein *Merge* fast nie so leicht über die Bühne. Häufig kommt es zu sogenannten *Merge-Konflikten*, da es ja sein kann, dass in verschiedenen Branches an verschiedenen Dateien gearbeitet wurde und *Git* nicht immer weiß, welche Änderungen denn jetzt bestehen bleiben sollen. Zwar ist das Werkzeug in der Lage, viele Konflikte selber zu lösen, aber eben nicht alle. Dann sind die Entwickler gefragt, und sie sitzen häufig zu dritt an einem Bildschirm und gehen Konflikt für Konflikt durch den Code, um alle Konflikte nach und nach zu lösen. Der richtige Spaß folgt dann bei Konfigurationsdateien für Tools, die ein Programm generiert ...

Haben Sie die Branches erfolgreich zusammengeführt, können Sie *intent_parsing* mit `git branch -d intent_parsing` löschen.

Commits rückgängig machen

Manchmal läuft die Entwicklung eines Features nicht wie gedacht und wir entscheiden uns, dass es besser ist, einen Schritt zurückzugehen und einen ganzen oder teilweisen Commit rückgängig zu machen. Zuerst können wir uns eine Commit-Historie über `git log` ausgeben lassen. Diese zeigt dann untereinander, welche Commits gemacht und wie diese kommentiert wurden, und vor allem, welche ID diese Commits hatten, denn jeder bekommt eine eindeutige Identifikationsnummer.

Mit dieser recht langen Identifikationsnummer können wir nun über `git show 1234` alle Änderungen in Commit 1234 anzeigen lassen. Nehmen wir mal an, wir hätten an der *main.py* gearbeitet und möchten diese nun zurückdrehen, dann würden wir das mit `git checkout 1234 main.py` erreichen.

Einen ganzen Commit rückgängig machen (also nicht nur eine einzelne Datei) ist ebenfalls möglich, nämlich über den Befehl `git revert HEAD`. *HEAD* referenziert den letzten Commit und kann durch eine beliebige Commit ID ersetzt werden.



TIPP: Haben Sie Ihre Änderungen noch nicht gepusht und möchten nur den Commit aus der lokalen Historie bekommen, genügt der Befehl `git commit --amend`.

■ 2.5 Die Komponenten unserer Anwendung

In diesem Abschnitt möchte ich Ihnen grob vorstellen, wie der Aufbau eines Sprachassistenten aussehen kann. Beginnen wir mit dem Block-Sprachassistent in der Mitte von Bild 2.5. Eine zentrale Aufgabe ist hier das Entgegennehmen von Sprachbefehlen, wozu wir einmal eine Audioeingabe sowie eine Funktion für die Umwandlung von Sprache in Text (*Text-To-Speech*) implementieren werden. Zuvor muss jedoch ein Aktivierungswort (*Wake Word*) genannt worden sein, vorher wird keine Übersetzung von Sprache in Text stattfinden. Ist das geschehen, wird der gesprochene Befehl von der *Dialog Engine* interpretiert und an das *Intent Processing* weitergeleitet. Zuvor wollen wir jedoch noch einiges tun, um die Verwendung unseres Sprachassistenten sicherer zu machen. Es soll versucht werden, anhand der Stimme per *Speaker Identification* zu erkennen, wer dort gesprochen hat, und per Abfrage an das *User Management*, ob diese Person überhaupt die Berechtigung hat, den Befehl auszuführen.

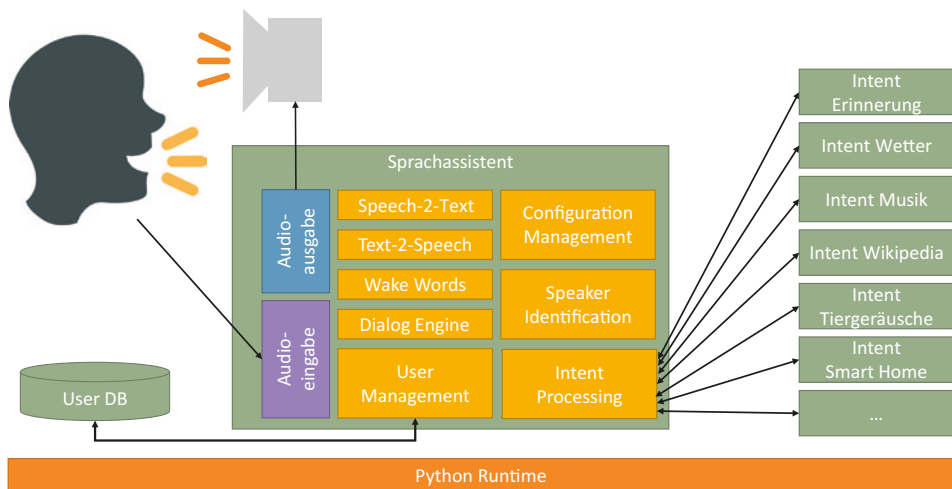


Bild 2.5 Der Aufbau der Anwendung gliedert sich in den Sprachassistenten, eine Datenbank zur Verwaltung von Benutzerdaten und eine Ansammlung verschiedener Intents.

Zuvor muss das *Intent Processing* jedoch zurückmelden, welcher Befehl für die Anfrage überhaupt in Frage kommt (was uns später noch in die Tiefen eines der Frameworks eintauchen lassen wird). Ist diese Autorisierung erfolgt, wird der passende Intent aufgerufen oder zurückgemeldet, dass der Intent nicht aufgerufen werden darf, oder dass kein passender Intent gefunden wurde. Nach einem erfolgreichen Aufruf erhält der Intent einige globale Konfigurationsparameter aus dem *Configuration Management* und führt seine Funktion aus. Eine solche Konfiguration kann etwa die Sprache sein, die wir für den Assistenten gewählt haben.



HINWEIS: Bild 2.5 zeigt, dass der gesamte Aufbau unserer Anwendung auf einer einzigen Python-Umgebung läuft. Wir werden uns später in Abschnitt 5.5 noch einen Microservice-Ansatz anschauen, der die Intents als eigenständige Anwendungen zu betreiben erlaubt.

Zu guter Letzt wird die Rückgabe des Intents, sollte er denn eine haben, über eine Text-To-Speech-Engine zurück in Sprache umgewandelt und durch die Lautsprecher ausgegeben. Moment, kann es denn Intents geben, die keine Rückgabe haben? Klar, die bekanntesten Vertreter sind entweder Unterbrechungsbefehle wie *Stop* oder Lautstärkeänderungen. Auch das Abspielen von Streams oder Musik hat nicht immer eine anschließende Sprachausgabe zur Folge.

■ 2.6 Erstellen eines Klassengrundgerüsts

Wir werden nun ein Grundgerüst erstellen, das als minimale Basis für unsere Anwendung dient. Dabei möchte ich einige Basics wiederholen, die dem einen oder anderen zu trivial erscheinen mögen. Tippen Sie in dem Fall einfach den Code ab und erfreuen Sie sich an den gewonnenen zwanzig Minuten.

Installation von Anaconda

Wir werden Anaconda verwenden, um unsere virtuelle Umgebung zu verwalten. Dazu müssen wir es zunächst installieren. Laden Sie sich die aktuelle Version der *64-Bit Individual Edition* von www.anaconda.com herunter und installieren Sie diese. Zum aktuellen Zeitpunkt ist das Version *2021.05*. Nach der Installation sollten Sie in Ihrem Startmenü den Eintrag *Anaconda3 (64-Bit)* vorfinden und darunter den *Anaconda Prompt*. Mit diesem werden wir viel Zeit verbringen, ein Shortcut lohnt sich also. Öffnen Sie diesen und gehen Sie in ein Verzeichnis, das Sie für die weitere Entwicklung Ihres Sprachassistenten verwenden möchten. Ich habe zwar im Beispiel-Repository für jedes Beispiel einen Unterordner angelegt, das müssen Sie aber nicht tun. Ein Ordner *Sprachassistent* oder eine entsprechende Abkürzung in Ihrem Benutzerverzeichnis reicht völlig.

Im *Anaconda Prompt* werden Sie sehen, dass Ihrem Pfad ein Umgebungsname in Klammern vorangestellt ist. Die Umgebung (*base*) ist die Basisumgebung, in der wir nicht arbeiten sollten. Legen Sie also Ihr erstes Environment bzw. Ihre erste Umgebung an:

```
conda create -n 001_hello_world python=3.8
```

Bestätigen Sie die Nachfrage, ob die benötigten Pakete heruntergeladen werden dürfen, mit *y*. Sie werden merken, dass Anaconda, wie angekündigt, keinen Ordner anlegt. Schauen Sie aber in Ihr Benutzerverzeichnis und dann in den Ordner *Anaconda3\envs*, dann werden Sie einen Ordner *001_hello_world* vorfinden, der alle Dateien beherbergt, die Sie ab der Aktivierung in Ihre Umgebung installieren.

Aktivierung ist ein gutes Stichwort, denn das werden wir zunächst tun.

```
conda activate 001_hello_world
```

Nun sollte sich der Umgebungsname von (*base*) in (*001_hello_world*) ändern, was zeigt, dass der Wechsel geklappt hat. Um wieder in die Umgebung (*base*) zurückzukehren, tippen Sie einfach `conda deactivate`. Das soll aber hier erst einmal nicht wieder passieren.

Legen Sie nun noch einen Arbeitsordner mit Namen *001_hello_world* im Verzeichnis *Sprachassistent* an und wechseln Sie mit `cd 001_hello_world` dort hinein.

Einstieg in die Entwicklung

Endlich ist es so weit, wir beginnen mit der Entwicklung. Öffnen Sie die IDE oder den Editor Ihrer Wahl. Ich möchte, so weit es geht, objektorientiert arbeiten. Das heißt, dass wir unseren Code in logisch unterteilte Klassen gliedern und mit Klassenfunktionen und -variablen versehen. Wir beginnen damit, eine Klasse *VoiceAssistant* zu erstellen.



TIPP: Manchmal steht hinter einem Klassennamen eine Klammer (`()`), manchmal nicht. Tatsächlich benötigt man diese Klammern, wenn die Klasse von einer anderen Klasse erben soll, sprich auf deren Struktur und Funktionsweise aufbauen soll. Leere Klammern waren in Python 2.x noch notwendig (ebenso wie die Angabe, dass eine Klasse immer von *object* erben sollte), in Python 3.x kann man die Klammern aber weglassen, wenn die Klasse nicht von einer anderen erbt.

Im Folgenden erstellen wir eine Methode `run()`, die die Logik unseres Sprachassistenten ausführen soll (siehe Listing 2.1). Diese wird uns über das ganze Buch hinweg begleiten. Die Definition einer Methode wird mit *def* eingeleitet und kann beliebig viele Parameter enthalten. Da es sich um eine Klassenmethode handeln soll, versehen wir sie mit dem Parameter *self*, der die Instanz der Mutterklasse referenziert.



HINWEIS: Der Begriff Instanz ist bekannt? Eine Klasse ist eine Art Schablone, wie der Bauplan eines Autos. Die Instanz einer Klasse ist dann das Auto, das bei Ihnen auf dem Hof steht, also etwas, das ganz genau dem Bauplan entspricht und die Funktionen ausführen kann, die im Plan genau beschrieben sind.

Die Definition der Methode `run()` muss eingerückt werden – wie, bestimmen Sie! Entweder per Tab oder mit einigen Leerzeichen. Wichtig ist nur, dass Sie sich konsequent an die Einrückung halten, denn diesbezüglich ist der Compiler sehr streng. Geschweifte Klammern wie in Java oder C# gibt es in Python nicht. Das PEP 8 (*Python Enhancement Proposal*) empfiehlt jedoch die Verwendung von vier Leerzeichen.

Listing 2.1 Definition einer Klasse für den Sprachassistenten

```

1. class VoiceAssistant:
2.     def run(self):
3.         print("Los geht's!")
4.
5. if __name__ == "__main__":
6.     va = VoiceAssistant()
7.     va.run()
```