Empfohlene Unterrichtsinhalte als Vorbereitung für dieses Thema: ***alle!***

12-02: Ausblick

Heute werden wir einige Themen besprechen, die wir hier noch nicht behandeln konnten, die euch aber vielleicht später über den Weg laufen werden... Viele dieser Themen werden übrigens im Kurs ***Advanced Natural Language Processing mit Python*** von Esther Seyffarth und Tatiana Bladier im Sommersemester 2021 behandelt.

Nachinstallation von Pythonmodulen und virtuelle Python-Umgebungen

Hier im Kurs haben wir nur mit Modulen gearbeitet, die zu jeder Standardinstallation von Python dazugehören (z.B. `random` oder `re`). Es gibt aber noch eine riesige Menge zusätzlicher Pythonmodule, die man sich separat installieren kann, wenn man bestimmte Aufgaben lösen und nicht den ganzen Code dafür selbst schreiben möchte. Pythonmodule kann man entweder vom *Python Package Index (PyPI)* beziehen oder sich von einer Plattform wie GitHub herunterladen.

Wenn wir zusätzliche Module installieren, sollten wir ein Konzept zur Verwaltung unserer Installationen haben, damit wir nicht durcheinanderkommen. Die gängigste Lösung dafür sind **virtuelle Python-Umgebungen**. Dabei erstellt man für jedes neue eigene Pythonprojekt eine minimale Python-Installation und ergänzt dort nur genau die Module, die man verwenden möchte. So ist sichergestellt, dass man mit einer stabilen Umgebung arbeitet und alle Module in ihren jeweiligen Versionen zueinanderpassen.

Das Tool, um Pakete nachzuinstallieren, heißt **pip**. Das Tool, um virtuelle Pythonumgebungen zu erstellen, heißt **venv**.

Objektorientierung

In diesem Kurs haben wir mit eingebauten **Datentypen** von Python gearbeitet: Strings, Listen, Dictionarys und so weiter. Dabei hatte jeder Datentyp eine Reihe von Methoden, die genau für diesen Datentyp verwendet werden können. Das ist sinnvoll, weil z.B. die Stringmethode `islower()` für Strings eine informative Aussage hat, die gleiche Methode aber auf Listen oder Dictionarys gar nicht angewendet werden soll.

Mit objektorientierter Programmierung haben wir die Möglichkeit, **eigene Klassen** zu definieren, die dann wie die eingebauten Datentypen ihre eigenen Methoden haben können. Außerdem kann man den Objekten, die zu einer selbstdefinierten Klasse gehören, eigene Attribute (also Eigenschaften) geben und diese im Laufe des Programmes verändern. Eine Klassendefinition ist ein "Rezept" für den Aufbau von Objekten. Das ist vergleichbar mit Funktionen: Eine Funktion ist ein "Rezept" für die Ausführung einer Teilaufgabe und kann mehrmals aufgerufen werden.

Mit Objektorientierung ist es noch leichter, die Funktionalität eigener Programme in Sinnabschnitte zu kapseln, die separat voneinander getestet werden können.

Test-driven development

Getestet werden? Na klar, bei größeren Projekten sollten wir immer testen, ob alles so funktioniert, wie wir uns das gedacht haben. Einen Vorgeschmack davon habt ihr schon bei den automatischen Tests in Moodle gesehen.

Beim **Test-driven Development** gehört das Schreiben automatischer Tests zum Entwickeln dazu. Die Idee ist, dass die Tests uns die Richtung angeben, sie werden also sogar als allererstes geschrieben!

Die Tests haben dann die Form von einer Art **Meta-Programm**, das unser eigentliches Programm aufruft und überprüft, ob für vorgegebene Eingaben die richtigen Ergebnisse erzeugt werden. Wir beginnen mit dem ersten Test und rufen ihn auf. Er schlägt fehl, weil unser eigentliches Programm noch gar nicht implementiert ist.

Als nächstes schreiben wir ein Programm, das **genau den Test** abdeckt, den wir geschrieben haben. Wenn unser Code dafür fertig ist, rufen wir den Test noch einmal auf. Wenn er erfolgreich ist, schreiben wir den nächsten Test oder erweitern den aktuellen Test.

Der Vorteil bei dieser Herangehensweise ist, dass wir automatisch mitbekommen, wenn Änderungen in unserem Projekt an irgendeiner Stelle einen Fehler erzeugen. Wir **checken kontinuierlich**, ob immer noch alle Tests erfolgreich sind, und fügen nach und nach mehr Tests und mehr Teile des Programms hinzu. Dadurch können wir jederzeit sicher sein, dass das Programm genau die Funktionen abdeckt, die durch unsere Tests explizit geprüft werden.

In [8]:
```
# Diese Funktion soll testen, ob für die Eingabe "Moby Dick"
# das Ergebnis "moby dick" von der Funktion ausgegeben wird.
def test_kleingeschrieben():
    assert kleingeschrieben("Moby Dick") == "moby dick", "Falsches Ergebnis für Moby
    assert kleingeschrieben("Ishmael") == "ishmael", "Falsches Ergebnis für Ishmael!
    print("Die Funktion erfüllt alle Tests.")

# Wir schreiben die Funktion so, dass sie den Test
# erfüllt.
def kleingeschrieben(eingabewort):
    return eingabewort.lower()

test_kleingeschrieben()
```

```
Die Funktion erfüllt alle Tests.
```

Umfangreiche Pythonpakete für Sprachverarbeitung (nltk, spacy) und Data Science (numpy, pandas, scikit-learn)

Wir sind nicht die ersten, die **computerlinguistische Analysen mit Python** durchführen wollen. Deshalb gibt es ganze Pakete, die man sich nachinstallieren kann und die umfangreiche sprachliche Analysen für uns durchführen können.

NLTK und **spaCy** sind aktuell die wichtigsten Pakete dieser Art für Sprachverarbeitung. Sie bringen nicht nur Pythonmodule und Funktionen mit, sondern auch Beispieldaten (z.B. Reden

amerikanischer Präsidenten, oder verschiedene Bibelübersetzungen) und vortrainierte statistische Modelle (z.B. zur intelligenten Zerlegung von Texten in einzelne Sätze).

Genau wie die computerlinguistischen Analysen, die wir von den Paketen **NLTK** und **spaCy** erledigen lassen können, sind auch statistische Untersuchungen eine oft gebrauchte Ressource. Dafür können wir Pakete wie **numpy**, **pandas** und **sklearn** verwenden. Diese Pakete erlauben uns, zum Beispiel Klassifizierer zu schreiben, die wir nur noch mit unseren eigenen Eingabedaten füttern müssen.

Mit diesen Paketen zu arbeiten ist etwas herausfordernd, wenn man sie noch nicht kennt, aber wenn man sich etwas damit beschäftigt, lohnt es sich fast immer, die darin definierten Module und Funktionen zu benutzen statt eine Aufgabe per Hand selbst zu lösen.

Best practices: Design patterns, code smells, Linting

In diesem Kurs lag der Fokus darauf, euch die Grundlagen von Python zu vermitteln und anhand von Beispielen zu üben, wie die verschiedenen Konzepte zusammengefügt werden können, um größere Aufgaben zu bearbeiten. Wenn man eine Programmiersprache frisch gelernt hat, ist man froh, wenn man eine Aufgabe *irgendwie* lösen kann. Wenn man etwas sicherer im Umgang mit der Sprache ist, ist es empfehlenswert, sich Gedanken über **guten Programmierstil** zu machen.

Hierfür finden sich online viele **Ressourcen, Leitfäden, vorgeschlagene Formatierungsregeln** und so weiter, die man alle in den eigenen Stil einfließen lassen kann. Dabei soll unser Code nicht nur lesbarer werden, sondern wir wollen auch versuchen, möglichst so zu programmieren, dass wir nicht später auf unerwartete Probleme stoßen.

Außerdem: Versionierungssoftware, Linux-Grundlagen, Bash

In fortgeschrittenen Programmierkursen wird es immer weniger um Elemente einer konkreten Programmiersprache gehen und immer mehr um den Workflow im Kontext bestimmter Projekte. Dazu gehören dann auch Themen wie Organisation im Team, der Umgang mit Linux und der dazugehörigen Skriptsprache bash, und andere allgemeinere Fähigkeiten, die unseren Werkzeugkoffer erweitern.

Zusammenfassung und Ausblick

Wenn ihr Spaß an diesem Pythonkurs hattet und gerne direkt noch mehr über Python lernen wollt, meldet euch gerne für den Kurs *Advanced Natural Language Processing mit Python* bei Esther Seyffarth und Tatiana Bladier im Sommersemester 2021 an. Der Kurs wird online stattfinden und auch sonst ähnlich wie der Kurs ablaufen, den wir hier gerade beenden.

Wenn ihr erst einmal eine Programmierpause braucht, könnt ihr den weiterführenden Kurs natürlich einfach später belegen. Wenn ihr Computerlinguistik studiert, begegnet euch Python in den nächsten Semestern sowieso immer wieder.

Wir freuen uns, dass ihr so erfolgreich an diesem Kurs teilgenommen habt, und wünschen euch alles Gute für euer weiteres Studium!

Und zum Schluss... eine Party!

Ihr seid alle herzlich zur **Institutsparty am 18. Februar 2021** eingeladen. Die Party wird in gather.town stattfinden, einer Plattform, in der man sich videospielartig durch einen 2D-Raum bewegt und einen Videocall mit anderen Personen startet, wenn die Avatare sich einander annähern. Eine Einladung mit Details erhaltet ihr kurz vorher.

Institutspartys sind typischerweise ein Ort, an dem sich Studierende, Dozierende und andere Mitarbeitende aus dem Institut in einem informellen Rahmen begegnen können. Wir würden uns sehr freuen, wenn ihr die Gelegenheit wahrnehmt und am 18. Februar mal vorbeischaut!