Teil 3: Datenbanken

Jarka Arnold
Aegidius Plüss

Datenbanken dienen der strukturierten Aufbewahrung grosser Datenmengen, die auch von mehreren Anwendern gemeinsam benutzt werden können.

INHALT

TEIL 3: DATENBANKEN

Version 1.0, August 2017

Autoren: Jarka Arnold, Aegidius Plüss

Kontakt: help@tigerjython.com

1. DIGITALISIERUNG VON INFORMATION

DU LERNST HIER...

wie man als Text vorliegende Informationen in elektronisch speicherbare Form umwandelt und was Bits und Bytes sind. Die Menschheit hat mehrere Jahrtausende vor Christus die Schrift erfunden, um Wissen festzuhalten, dass dieses später wieder verfügbar war, aber auch um Informationen anderen Personen übermitteln zu können. Statt Steintafel und Papier wird heute Wissen und Information in elektrische Signale umgewandelt, die lediglich zwei Werte, 0 und 1 enthalten. Man nennt diese Elementarinformation ein Bit.

Mit einem Bit lassen sich 2 Zeichen unterscheiden, mit 2 Bits 4 Zeichen, mit 3 Bits 8 Zeichen usw..

0	1

1 Bit

00	01	10	11

4 Zeichen mit 2 Bits

000	001	010	011	100	101	110	111

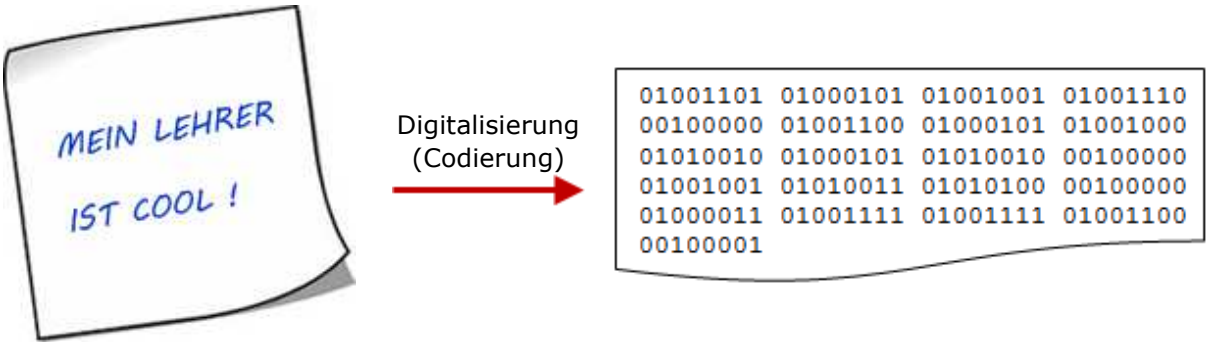
8 veschiedene Zeichen mit 3 Bits

Mit 8 Bits (1 Byte) kann man 256 verschiedene Zeichen darstellen. Damit kann das das ganze Alphabet mit Gross- und Kleinbuchstaben inklusive Umlaute und einige Sonderzeichen codiert werden.

MUSTERBEISPIEL

Bei der Codierung von Texten wird in der Regel der ASCII-Code verwendet. Dieser ordnet jedem Zeichen des Texts 1 Byte = 8 Bits zu. So wird z.B. der Buchstabe A als 0100001 codiert.

In einem Python-Programm kannst du die Funktion *strToBin(c)* verwenden. Diese liefert zu einem Zeichen c den zughörenden Code (als Zeichenkette mit 0 und 1).



Mit deinem Programm kannst du einen beliebigen Text eingeben, der dann codiert als Bytefolge ausgeschrieben wird.

```
text = inputString("Gib einen Text ein")
print text
for buchstabe in text:
    print strToBin(buchstabe)
```

MERKE DIR...

Textinformationen werden elektronisch als 0/1-Werte (bits) codiert. Oft verwendet man für einen Buchstaben 8 Bits = 1 Byte. Der internationale Standard für diese Codierung nennt man ASCII-Code.

ZUM SELBST LÖSEN

1a. Du gibst in einem Eingabedialog einen Text ein. Schreib aus, wieviele Bytes er in digitalisierter Form umfasst.

1b. Ein durchschnittlicher Roman umfasst 100'000 Wörter. Schätze ab, welcher Speicherbedarf (in Bytes oder kBytes) dazu nötig ist.

1c. Aktuell kannst du auf den grössten USB-Memorysticks (Flash drives) bis zu 1 Terrabyte (1 TB) speichern. Orientiere dich, was ein Terrabyte ist und schätze ab, viele Romane darauf Platz finden.

2. Mit der Funktion *binToStr(b)*kannst du aus einem Byte (in der Schreibweise "01000001") den Buchstaben zurück erhalten, beispielsweise schreibt

```
b = "01000001"
print binToStr(b)
```

den Buchstaben A aus. Eine Freundin übermittelt dir ihren Namen in digitalisierter Form als Liste

```
["01000001", "01001110", "01000100", "01010010", " 01000101", "01000001"]
```

Finde ihren Namen heraus.

3a. Die Digitalisierung führt meist zu einem bestimmten Verlust an Information. Überlege dir, welche Information verloren geht, wenn man einen handgeschriebenen Brief in digitalisierter Form (beispielsweise als Email) überträgt.

3b. Begründe, warum man durch die Verwendung von Emoticons den Informationsverlust durch die Digitalisierung etwas verkleinern kann.

2. WOZU DATENBANKEN?

DU LERNST HIER...

wie du Datenbanken als digitale Informationsspeicher verwenden kannst und das Computersysteme mehr als nur Rechenmaschinen sind, denn sie können Daten gemäss bestimmter Kriterien millionenfach schneller verarbeiten als der Mensch und die Grösse des Datenspeichers ist praktisch unbegrenzt.

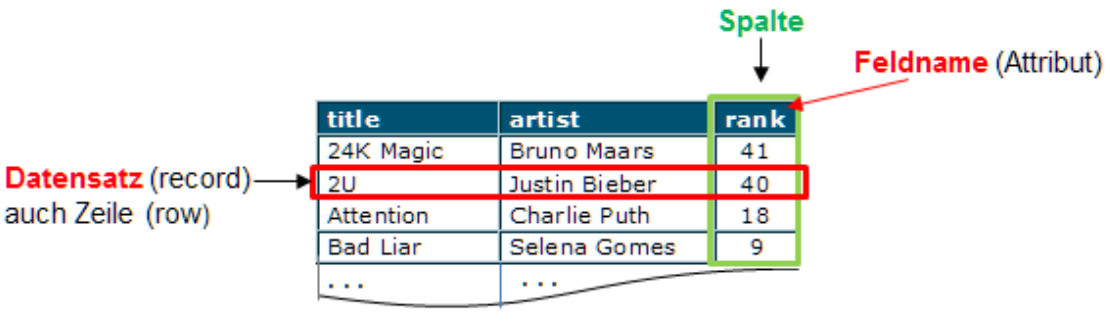
Da in unserer digitalen Gesellschaft täglich enorme Mengen von Information in Datenbanken gesammelt und weiter verarbeitet werden, sind grundlegende Kenntnisse über Datenbanken auch für dich wichtig, damit du beispielsweise verstehst, wie soziale Medien und personalisierte Werbung funktionieren, und wie du dich vor dem Missbrauch von deinen eigenen Personendaten schützen kannst.

MUSTERBEISPIEL

Viele Informationen werden in Form von Tabellen strukturiert. Gegenüber einer Beschreibung als Text können die so strukturierten Daten viel effizienter gespeichert und weiter verarbeitet werden. Aus diesem Grund sind Tabellen das Basiselement vieler Datenbanken.

Im folgenden typischen Beispiel werden Titel und Künstler von aktuell weltberühmten Songs zusammen mit ihrem Rang in einer Hitliste (eines Anbieters von Musik-Downloads) erfasst. Es könnte sich auch um eine Mediathek deiner eigenen Songs handeln.

Eine Tabelle besteht aus einer gitterartigen Anordnung von Zeilen (rows) und Spalten (columns). Jede Zeile ist eine Informationseinheit, die Datensatz (record) genannt wird. Der Datensatz enthält die Felder mit den Datenwerten. Die Spalten haben einen eindeutigen Feldnamen (auch Attribut genannt). Der Beginn der Tabelle *topsongs* sieht also so aus:



In der Distribution von TigerJython befinden sich in der Datenbank *tigerjython.db* mehrere nützliche Tabellen. Du kannst die Tabellen und Feldnamen (Attribute) mit folgendem Programm anzeigen:

```
from dbtable import *

showDbInfoTJ("tigerjython.db")
```

Um die Tabelle *topsongs* zu verwenden, musst du das Modul *dbtable* importieren und dann ein (leeres) Tabellenobjekt erzeugen. Nachher holst du die Tabellendaten aus der Datenbank *tigerjython.db* und stellst die Tabelle mit *print* im Ausgabefenster dar.

```
from dbtable import *

topsongs = DbTable()
topsongs.restoreFromTJ("tigerjython.db")
print topsongs
```

Ergebnis (hier nur 4 von den angezeigten 100 Zeilen):

title	artist	rank
24K Magic	Bruno Mars	41
2U	Justin Bieber	40
Attention	Charlie Puth	18
Bad Liar	Selena Gomez	9

Oft möchte man eine Tabelle nach einem der Felder sortiert ausschreiben. Ohne Mühe kannst du die Tabelle nach der Bewertung sortiert ausschreiben.

```
from dbtable import *

topsongs = DbTable()
topsongs.restoreFromTJ("tigerjython")
topsongs.sort("rank")
print topsongs
```

Du kannst die Tabelle auch in einer for-Schleife zeilenweise durchlaufen und auf die Felder mit dem Punktoperator zugreifen.

Beispielsweise schreibst du mit dem folgenden Programm nur die 5 höchstbewerteten Titel aus.

```
from dbtable import *

topsongs = DbTable()
topsongs.restoreFromTJ("tigerjython")
topsongs.sort("rank")
for record in topsongs:
    if record.rank <= 5:
        print record.rank, record.title
```

Im Ausgabefenster siehst du:

1 Despacito
2 Believer
3 Wild Thoughts
4 Slow Hands
5 Body Like a Back Road

MERKE DIR...

Tabellen sind Strukturierungsmittel, damit Daten effizient in einer Datenbank gespeichert, nach bestimmten Kriterien leicht weiter verarbeitet und anschaulich dargestellt werden können. Im Gegensatz zu Variablen sind die Daten in einer Datenbank dauerhaft (oder persistent), "überleben" also die Programmdauer und können jederzeit später wieder verwendet werden. Unter der Datenbank versteht man meist die Datenbankdatei zusammen mit den zugehörenden Programmierwerkzeugen, um die Daten zu manipulieren.

ZUM SELBST LÖSEN

1. Mit Datenbanken kannst du auch grosse Datenmengen sehr effizient untersuchen. Stellst du dir die Frage, welche Songs von Justin Bieber gesungen werden, müsstest du ohne Computer mühsam alle 100 Songs einzeln durchsehen. Schreibe ein Programm, das diese Arbeit für dich erledigt und die Titel von Justin Bieber im Ausgabefenster auschreibt.

2a. Du kannst mit *mountains.restoreFromTJ("tigerjython.db")* in der Tabelle *mountains* Informationen über die höchsten Berge in verschiedenen Ländern erhalten. Schreibe die Tabellendaten im Ausgabefenster aus.

2b. Schreibe die Berge sortiert nach der Höhe aus. Meist möchte man allerdings die Sortierreihenfolge umkehren, damit zuerst die höchsten Berge erscheinen Verwende dazu den Befehl

```
mountains.sort("height", False)
```

2c. Schreibe alle Schweizer Berge aus. (Du kannst auch eine ganze Zeile mit *print record* ausschreiben.)

2d. Schreibe alle Schweizer Berge sortiert nach der Höhe aus, die höher als 3000 m sind.

3. TABELLEN ERSTELLEN

DU LERNST HIER...

wie man Informationen in einer Datenbank dauerhaft abspeichern kann und wie man später wieder darauf zurückgreift. Dazu erstellst du eine Tabelle mit Personendaten, fügst Datensätze hinzu und speicherst die Tabelle in einer Datenbank.

MUSTERBEISPIEL

Du möchtest die Personendaten deiner Klassenkameraden in einer Datenbank abspeichern. Dabei verwendest du Namen, Vornamen, Wohnort, Geschlecht und Jahrgang. Als erstes erstellst du eine Tabelleninstanz von *DbTable*, wo du die Feldnamen festlegst. Gewöhnlich wird ein vielsagender Feldname gewählt, der allerdings meist klein geschrieben wird und sich an die Regeln für Variablenamen halten muss, also keine Umlaute und Spezialzeichen enthalten darf. In Personentabellen ist es üblich, ein zusätzliches Feld *id* mit ganzen Zahlen 1, 2, 3. ... aufzunehmen, damit die Personen über diese Zahl eindeutig identifiziert werden können.

Mit dem Befehl *insert()* fügst du einzelne Personen in die Tabelle, wobei die Reihenfolge der Parameter derjenigen der Felder entsprechen muss. Du kannst natürlich auch andere Personen einfügen, z.B. dich selbst in die Tabelle aufnehmen. Nach dem Einfügen schreibst du die Tabelle zur Kontrolle im Ausgabefenster aus.

Am Schluss speicherst du die Tabelleninformationen in einer Datenbank mit dem Namen *schule.db*, die beim Abspeichern mit *save()* erzeugt wird und sich im gleichen Verzeichnis wie dein Programm befindet.

```
from dbtable import *

persons = DbTable('id', 'name', 'vorname', 'wohnort', 'geschlecht', 'jahrgang')
persons.insert(1,'Huber','Lia','Bern','w', 2002)
persons.insert(2,'Meier','Luca','Basel','m', 2003)
persons.insert(3,'Frech','Tim','Bern','m', 2000)
persons.insert(4,'Bauer','Jan','Luzern','m', 2003)
persons.insert(5,'Zwahlen','Noah','Thun','m', 2002)
persons.insert(6,'Meier','Nina','Biel','w', 2001)
print persons
persons.save('schule.db')
```

Ergebnis:

id	name	vorname	wohnort	geschlecht	jahrgang
1	Huber	Lia	Bern	w	2002
2	Meier	Luca	Basel	m	2003
3	Frech	Tim	Bern	m	2000
4	Bauer	Jan	Luzern	m	2003
5	Zwahlen	Noah	Thun	m	2002
6	Meier	Nina	Biel	w	2001

Mit einem Dateimanager kannst du überprüfen, dass im Programmverzeichnis tatsächlich eine Datei *schule.db* entstanden ist.

Wesentliches Merkmal von Datenbanken ist, dass die Daten nach Ende des Programms dauerhaft abgespeichert sind Du kannst also den Computer abschalten und später wieder auf deine Personendatenbank *schule.db* zugreifen. Du kannst diese Datei sogar auf den Computer eines Klassenkameraden kopieren und er kann die Daten dort verwenden. Dazu verwendest du den Befehl *restore(),* um die Daten aus der Datenbank wieder in ein DbTable-Objekt zurück zu holen. Du musst als Variablenname *person* wählen, da in der Datenbank die Tabelle mit diesem Namen abgelegt wurde.

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
print persons
```

MERKE DIR...

Beim Erzeugen einer Tabelle müssen die Feldnamen festgelegt werden. In den Feldern können nur ganze Zahlen (int), Dezimalzahlen (float) und Zeichenketten (str) verwendet werden. Später wirst du auch lernen, wie man Bilder abspeichern kann. Mit save() speicherst du die Tabellendaten in der angegebenen Datenbank ab, mit restore() kannst du Daten jederzeit später mit einem anderen Programm wieder herstellen.

Natürlich kannst du die Datei *schule.db* auch auf einen anderen Computer kopieren und von dort mit einem Programm im gleichen Verzeichnis auf die abgelegten Daten zurückgreifen.

ZUM SELBST LÖSEN

1. Füge in die Tabelle *person* zwei weitere Datensätze ein. Zum Beispiel:

id	name	vorname	wohnort	geschlecht	jahrgang
8	Amsler	Fabio	Zürich	m	2005
9	Zürcher	Nina	Aargau	w	2003

2. Schreibe ein Programm, um die Personendaten wieder zu holen und sortiert nach dem Namen im Ausgabefenster auszuschreiben.

4. DATENSÄTZE AUSWÄHLEN (SELECT)

DU LERNST HIER...

wie du in einer Datenbank gezielt nach bestimmten Informationen suchst und nur ausgewählte Informationen anzeigst. Die gezielte Auswahl bestimmter Informationen aus einer riesigen Datensammlung, auch Datenreduktion oder Datenfilterung genannt, ist eine wichtige Datenbankoperation.

Bereits im zweiten Kapitel hast du gelernt, dass du mit *select()* Datensätze nach einem bestimmten Kriterium auswählen kannst. Da aber *select()* ein verschachteltes Tupel mit Datensätzen zurück gibt, brauchst es etwas Anstrengung, um die Informationen zu extrahieren.

MUSTERBEISPIEL

Du gehst wieder von der Personendatenbank aus, die du in den vorherigen Kapiteln erzeugt hast, interessierst du aber nur für den Namen, Vornamen und den Jahrgang von Mädchen. Dazu verwendest du den *select*-Befehl mit einer zusätzlichen Angabe, die festlegt, welche Felder du zurück erhalten willst. Folgender Aufruf liefert dir das Gewünschte:

```
persons.select("name", "vorname", "jahrgang", geschlecht = "w")
```

Das zurückgelieferte Tupel nennt man oft einen *ResultSet*, denn es handelt sich um eine Menge von günstig verpackten Datensätzen.

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
resultSet = persons.select("name", "vorname", "jahrgang", geschlecht = "w")
print resultSet
```

Ergebnis: `(('Huber', 'Lia', 2002), ('Meier', 'Nina', 2001))`

Am einfachsten kannst du den auf die einzelnen Datensätze des *ResultSets* mit einer for-Schleife

```
for row in resultSet:
```

zugreifen. Da *row* selbst wieder ein Tupel ist, erhältst du die Werte einzelner Felder mit einem Index: *row[0]* liefert den Namen, *row[1]* den Vornamen und *row[2]* den Jahrgang. Du kannst damit beispielsweise die Mädchen wieder in einer neuen Tabelle *girls* speichern.

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
resultSet = persons.select("name", "vorname", "jahrgang", geschlecht = "w")
print resultSet
girls = DbTable("name", "vorname", "jahrgang")
for row in resultSet:
    girls.insert(row[0], row[1], row[2])
print girls
girls.save("schule.db")
```

Mit dem folgenden Programm kannst du jetzt die Information zurückholen und anzeigen:

```
from dbtable import *

girls = DbTable()
girls.restore("schule.db")
print girls
```

MERKE DIR...

Der *select()*-Befehl liefert einen *ResultSet in einem Tupel*, in dem jeder Datensatz wieder in ein Tupel gepackt ist. Auf die einzelnen Felder greifst du mit einem Index zu. Im *select*-Befehl kann man leider Gleichheitsbedingungen angeben. Du kannst aber den *ResultSet* durchlaufen und dort nach einer Bedingung suchen. Beispielsweise findest du alle Teens (mit Jahrgang später als 2001 und früher als 2005 mit

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
resultSet = persons.select()
for row in resultSet:
    if row[5] > 2001 and row[5] < 2005:
        print row
```

ZUM SELBST LÖSEN

1. Hole alle Personen mit Wohnort Bern aus der Personendatenbank und füge sie in eine Tabelle *berner*, die du dann ausschreibst.

2a. Verwende die Tabelle mountains (aus der TigerJython-Distribution) und suche alle Schweizer "Viertausender".

2b. Speichere die Namen und die Höhe dieser Berge in einer Datenbank *hochgebirge* in der Tabelle *viertausender*.

3. Die Tabelle *countries* (aus der TigerJython-Distribution) enthält u.a. die Felder *country_de* mit dem deutschen Ländernamen und *continent* mit dem Kontinent. Schreibe im Ausgabefenster alle Ländernamen mit dem Kontinent "Africa" untereinander aus.

5. DATENSÄTZE MUTIEREN (UPDATE)

DU LERNST HIER...

wie du den Wert bestimmter Felder eines ausgewählten Datensatzes oder mehrerer Datensätze gleichzeitig ändern kannst. Statt wie bei Variablen von einer Zuweisung zu sprechen, sagt man bei Datenbanken, man *mutiere* die Daten oder man führe einen *Update* durch.

MUSTERBEISPIEL

Beim Mutieren handelt es sich eigentlich um einen zweistufige Operation, bei der du zuerst festlegen musst, welche Datensätze zu mutieren sind und dann s einem oder mehreren Feldern und neue Werte zuordnest. Dazu wir der Befehl *update()* in einer etwas ungewohnten Form verwendet, indem du wie bei *select()* mit einer Gleichheitsbedingung die Datensätze auswählst und sie nachher mit einer Zuweisungsoperation neu setzt. Da beide Angaben das gleiche Format haben, muss man zur Unterscheidung bei der Zuweisung dem Feldnamen einen Underline _ vorstellen.

Im Beispiel wechselt Bauer Jan seinen Wohnort nach Zürich.

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
print persons
persons.update(name = "Bauer", vorname = "Jan")(wohnort = "Zürich")
print persons
persons.save("schule.db")
```

Damit die Mutation der Tabelle tatsächlich in der Datenbank übernommen wird, musst du die Tabelle mit *save()* speichern.

MERKE DIR...

Wie bei *select()* wählst du mit Gleichheitsbedingungen die Datensätze aus, die du mutieren willst. Dies ist dann sehr gefährlich, wenn mehrere Datensätze die Bedingungen erfüllen, d.h. wenn es in diesem Beispiel mehrere Bauer Paul gibt. In diesem Fall werden alle Datensätze modifiziert, wie man leicht zeigen kann, wenn Meier Nina von Biel nach Aarau umzieht und man dies mit folgendem Programm unternimmt:

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
print persons
persons.update(name = "Meier")(wohnort = "Aarau")
print persons
```

Nun ist nicht nur Meier Nina, sondern auch Meier Luca nach Aarau umgezogen!

Selbst wenn man mehrere Felder als Auswahlbedingung angibt, ist man nie ganz sicher, ob nur ein bestimmter Datensatz ausgewählt wurde. Dies ist einer der Gründe, warum man üblicherweise ein zusätzliches Feld *id* einführt, über das man einen Datensatz eindeutig identifizieren kann. Dieses nennt man ein *Schlüsselfeld* oder kurz einen *Schlüssel* (key field, key)

ZUM SELBST LÖSEN

1. Meier Luca zieht von Basel nach St. Gallen um. Führe den Update unter Verwendung des Schlüssels durch.

2. Alle männlichen Personen ziehen in eine Wohngemeinschaft nach Zürich. Führe die Mutation in einem einzigen update()-Befehl durch.

6. DATENSÄTZE LÖSCHEN (DELETE)

DU LERNST HIER...

wie man einen, mehrere oder alle Datensätze einer Tabelle löschen kann. Dabei musst du aber sehr vorsichtig sein, damit du nicht versehentlich Daten verlierst.

MUSTERBEISPIEL

Da Zwahlen Noe an eine andere Schule übertritt, wird ihr Datensatz aus der Tabelle *persons* gelöscht. Dazu verwendest du den *delete*-Befehl, bei dem du wie beim *select*- und *update*-Befehl den betreffenden Datensatz mit Gleichheitsbedingungen auswählst. Erfüllen mehrere Datensätze die Bedingungen, so werden sie alle gelöscht. Aus diesem Grund verwendest du am besten das Schlüsselfeld *id*, das den Datensatz eindeutig festlegt. Du kannst die *id* durch Ausschreiben der Tabelle herausfinden. Wieder gehst du von der Tabelle *persons* aus, die du im Kapitel 3 in der Datenbank *schule.db* erzeugt hast.

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
print persons
persons.delete(id = 5)
print persons
persons.save("schule.db")
```

Damit die Veränderung der Tabelle tatsächlich in der Datenbank übernommen wird, musst du sie mit *save()* speichern.

MERKE DIR...

Beim Löschen von Datensätzen musst du auf die Eindeutigkeit achten, sonst verlierst du ungewollt gewisse Datensätze. Gibst du beispielsweise gar keine Bedingung an, so wählst du damit alle Datensätze aus und es werden dementsprechend alle gelöscht. Das folgende Programm zeigt dann eine leere Tabelle (die aber zum Glück nicht gespeichert wird).

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
print persons
persons.delete()
print persons
```

ZUM SELBST LÖSEN

1. Hole dir wie in Kapitel 2 mit *topsongs.restoreFromTJ("tigerjython.db")* die Top Songs und lösche alle Songs von Ed Sheeran. Zeige die Tabelle nachher sortiert nach Artistennamen an.

2a. Hole dir wie im Kapitel 2 mit *mountains.restoreFromTJ("tigerjython.db")* Informationen über die höchsten Berge. Zeige im Ausgabefenster alle Berge an, die weniger hoch als 3000 m sind.

2b. Lösche in der Tabelle alle Berge, die weniger hoch als 3000 m sind, und zeige die Tabelle an.

7. TABELLEN VERKNÜPFEN (JOIN)

DU LERNST HIER...

dass das Verknüpfen und Zusammenführen von Tabellen zu einem wesentlichen Informationsgewinn führt und die Datenbankmanipulationen vereinfacht und besser überschaubar macht.

Es ist davon auszugehen, dass sich jeder Bürger, so auch du, mit Namen, Vornamen und weiteren persönlichen Informationen in Hunderten von Datenbanken befindet und identifizierbar ist, beispielsweise auf Grund seiner Einkäufe mit einer Kundenkarte, der Beteiligung an Umfragen über gekaufte Produkte oder seinen Aktivitäten in sozialen Netzwerken. Kombiniert man diese Daten, so lässt sich ein Profil einer Person erstellen, das weit über die Informationen aus einer einzigen Datenbank hinausgeht und oft den Schutz der Privatsphäre verletzt.

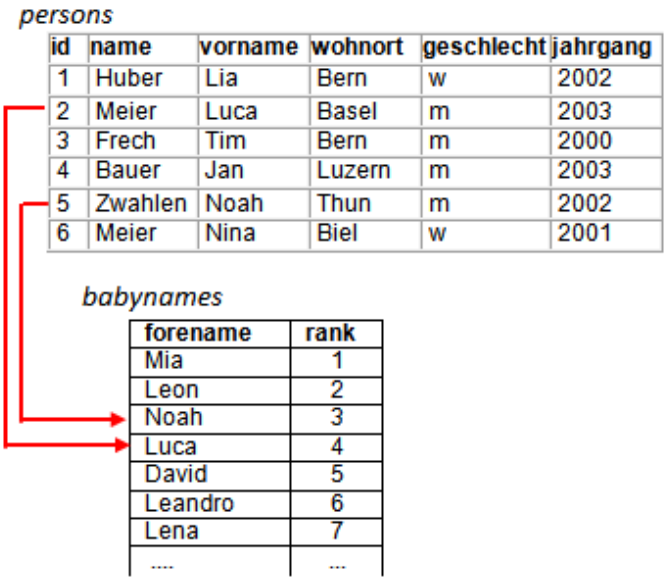
MUSTERBEISPIEL

Wir wollen eine Beliebtheitsliste bei der Wahl von Vornamen von Neugeborenen heranziehen und diese auf die Vornamen der Personentabelle *persons* deiner Klasse anwenden, um herauszufinden, welche Beliebtheitsreihenfolge diese Vornamen haben. Ist die Beliebtheitsliste relativ klein (hier enthält sie vorerst nur 100 Namen), so liesse sich dies zwar auch ohne Computer meistern. Wir werden aber nachfolgend die Reihenfolge aller Vornamen in der Schweiz verwenden, die über 50'000 Einträge enthält. Ein manuelles Vorgehen ist in diesem Fall undenkbar.

Zuerst stellen wir die beiden Tabellen im Ausgabefenster dar. (Die Tabelle der Vornamen beziehen wir aus der TigerJython-Distribution.)

```
from dbtable import *

persons = DbTable('id','name','vorname','wohnort','geschlecht','jahrgang')
persons.insert(1, 'Huber', 'Lia', 'Bern', 'w', 2002)
persons.insert(2, 'Meier', 'Luca', 'Basel', 'm', 2003)
persons.insert(3, 'Frech', 'Tim', 'Bern', 'm', 2000)
persons.insert(4, 'Bauer', 'Jan', 'Luzern', 'm', 2003)
persons.insert(5, 'Zwahlen', 'Noah', 'Thun', 'm', 2002)
persons.insert(6, 'Meier', 'Nina', 'Biel', 'w', 2001)
print persons

babynames = DbTable()
babynames.restoreFromTJ("tigerjython.db")
print babynames
```

Für die Lösung unserer Aufgabe geht es offenbar darum, die beiden Tabellen nach gleichen Vornamen zu untersuchen, also eine *Beziehung* (*Relation)* zwischen den Feldern *vorname* der Tabelle *persons* und *forename* der Tabelle *babynames* zu erstellen.



Um diese Beziehung zwischen den Feldern *name* und *forename* zu erstellen, werden die beiden Tabellen mit dem Befehl *persons.join(babynames)* miteinander verbunden. Dabei wird jeder Datensatz der Tabelle *persons* mit jedem Datensatz der Tabelle *babynames* kombiniert. Wir sind aber nicht an dieser grossen Tabelle (mit 6 x 100 = 600) Datensätzen interessiert, sondern nur an denjenigen, bei denen die Vornamen übereinstimmen. Wir setzen dazu die Feldnamen *persons.vorname* und *babynames.forename* für die Übereinstimmung in den *join*-Befehl.

```
from dbtable import *

persons = DbTable('id','name','vorname','wohnort','geschlecht','jahrgang')
persons.insert(1, 'Huber', 'Lia', 'Bern', 'w', 2002)
persons.insert(2, 'Meier', 'Luca', 'Basel', 'm', 2003)
persons.insert(3, 'Frech', 'Tim', 'Bern', 'm', 2000)
persons.insert(4, 'Bauer', 'Mia', 'Luzern', 'm', 2003)
persons.insert(5, 'Zwahlen', 'Olivia', 'Thun', 'm', 2002)
persons.insert(6, 'Meier', 'Nina', 'Biel', 'w', 2001)

babynames = DbTable()
babynames.restoreFromTJ("tigerjython.db")

table = persons.join(babynames, persons.vorname, babynames.forename, True)
table.sort("rank")
print table
```

Schon ist das Problem gelöst, denn im Ausgabefenster siehst du:

id	name	vorname	wohnort	geschlecht	jahrgang	forename	rank
1	Huber	Lia	Bern	w	2002	Lia	37
2	Meier	Luca	Basel	m	2003	Luca	4
3	Frech	Tim	Bern	m	2000	Tim	22
4	Bauer	Jan	Luzern	m	2003	Jan	21
5	Zwahlen	Noah	Thun	m	2002	Noah	3
6	Meier	Nina	Biel	w	2001	Nina	26

MERKE DIR...

Durch die Verbindung von Tabellen lassen sich viele neue Informationen gewinnen, die in den Einzeltabellen nicht vorhanden sind. Lässt man im join-Befehl den letzten Parameterwert True weg, wo werden die Vergleichsfelder nicht in die verbundene Tabelle aufgenommen.

ZUM SELBST LÖSEN

1. Anlässlich eines Sporttags deiner Schule erreichen die Personen aus der Tabelle persons folgende Leistungen in den Sportarten Lauf, Hochsprung und Weitsprung, die in der Tabelle *sport* abgelegt sind:

```
sport = DbTable('sid', 'lauf', 'hochsprung', 'weitsprung')
sport.insert(1, 11.2, 1.23, 4.15)
sport.insert(3, 14.1, 1.41, 3.92)
sport.insert(6, 12.6, 1.22, 3.88)
sport.insert(4, 11.5, 4.2, 1.20)
```

a. Verbinde die Tabellen *persons* und *sport* über *persons.id* und *sport.sid* und zeige die Tabelle an.

b. Sortiere die Tabelle, damit eine Rangliste auf Grund des Weitsprungs angezeigt wird.

2. Ergänze die Tabelle *persons* mit deinem Namen und verwende die Information aller rund 50'000 Vornamen in der Schweiz für die Untersuchung der Häufigkeit. Anleitung: Die Tabelle *firstnames* hat die Feldnamen: *fname* für den Vornamen und *fwoman, fman* für die Häufigkeiten für Männer bzw. Frauen. Es ist auf Grund der Grösse keine gute Idee, die Tabelle *firstnames* auszuschreiben, sondern nur die verbundene Tabelle.

8. KLIMADATENBANK

DU LERNST HIER...

wie du grosse Datenmengen aus dem Internet holst und sie für deine Zwecke verwendest. Dabei gehst du von der Fragestellung aus, wie sich das Klima in den letzten rund 100 Jahren in verschiedenen Ländern verändert hat. Du gehst dabei von den Jahresmittelwerten der Temperaturen von 1901 bis 2015 in 89 Ländern aus.

Die Rohdaten beziehst du von der CRU (Climate Research Unit) auf http://www.cru.uac.uk. Sie stehen dir aufgearbeitet in der Tabelle *meantemp* der Datenbank *tigerJython.db* zur Verfügung, die sich in der Distribution von TigerJython befindet.

MUSTERBEISPIEL

Die Tabelle *meantemp*, die sich in der Distribution von TigerJython befindet, hat das Feld *country* mit dem Ländernamen und Felder mit den Jahrzahlen *1901* bis *2015*für die mittleren Jahrestemperaturen in diesen Jahren. Du kannst zuerst mit folgendem Programm die Werte für alle Länder anzeigen, musst aber etwas Geduld haben, bis der Computer die Daten extrahiert hat.

```
from dbtable import *

meantemp = DbTable()
meantemp.restoreFromTJ("tigerjython.db")
print meantemp
```

Willst du nur die Daten für die Schweiz ausschreiben, so verwendest du den Befehl *view():*

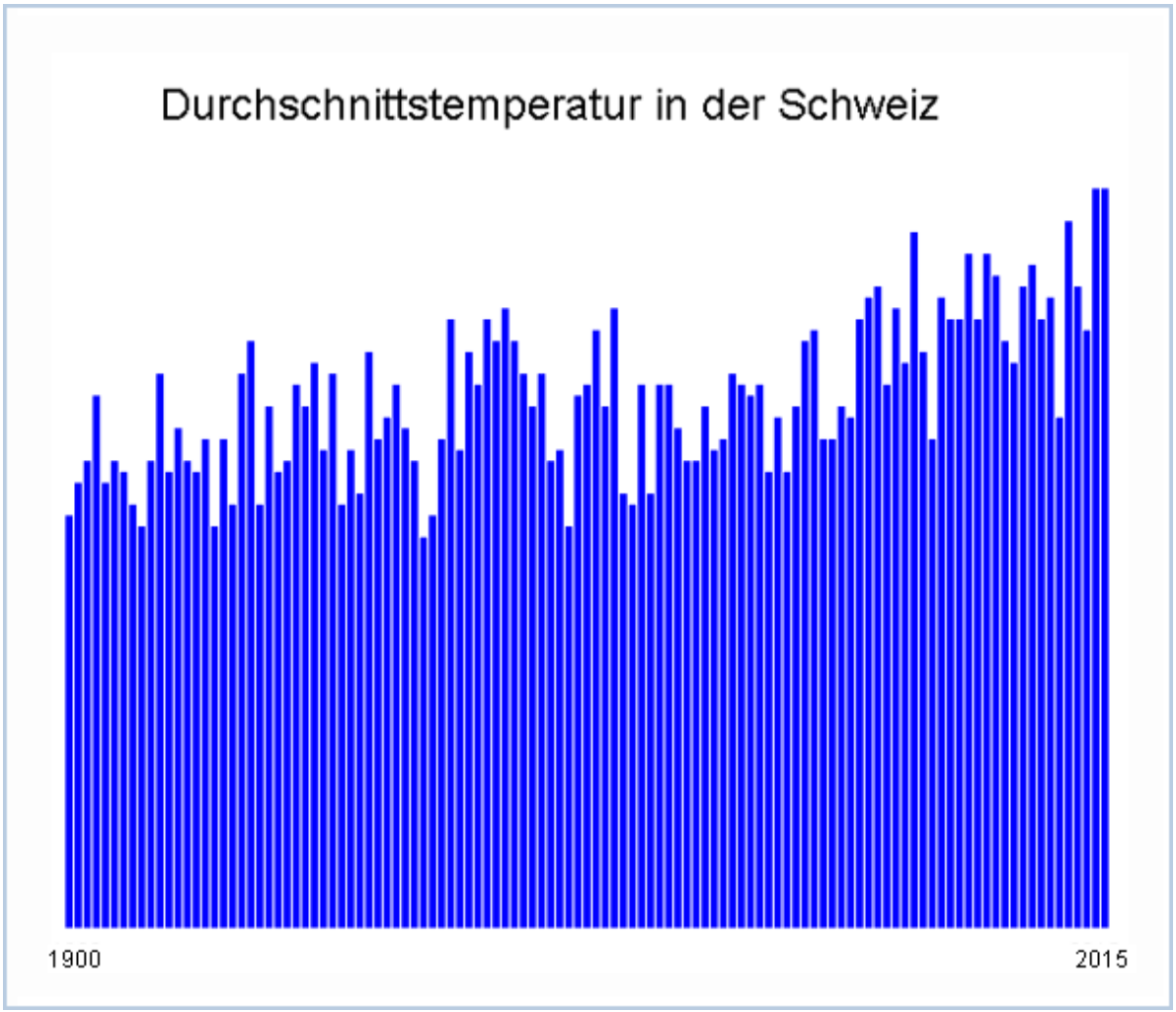
```
from dbtable import *

meantemp = DbTable()
meantemp.restoreFromTJ("tigerjython.db")
meantemp.view(country = "Switzerland")
```

Der Befehl *view()* funktioniert ähnlich wie *select()*, bezieht sich aber auf das Ausschreiben der Tabelle im Ausgabefenster. Mit *view(country = "Switzerland")* wählst du nur den Datensatz aus, der sich auf die Schweiz bezieht.

Viel schöner und eindrücklicher als eine Zahlenreihe ist eine grafische Darstellung. Die Turtlegrafik kann dir dabei behilflich sein, indem du mit der Turtleeinen Balken als dicke Turtlespur zeichnet.

```
from dbtable import *
from gturtle import *

def drawBar(s):
    left(90)
    penDown()
    forward(s)
    back(s)
    penUp()
    right(90)

makeTurtle()
```

```
ht()
right(90)
setLineWidth(4)
setPos(-290, -250)

meantemp = DbTable()
meantemp.restoreFromTJ("tigerjython.db")

resultSet = meantemp.select(country = "Switzerland")
value = resultSet[0]
for i in range(1, len(value)):
    print value[i]
    s = int(value[i] * 60)
    drawBar(s)
    forward(5)

setPenColor("black")
setPos(-240, 240)
label("Durchschnittstemperatur in der Schweiz")
setFontSize(12)
setPos(-300, -270)
label("1900")
setPos(260, -270)
label("2015")
```



In der Grafik ist die Temperaturzunahme offensichtlich.

MERKE DIR...

Datenbanken sind besonders für grosse Datenmengen hervorragend geeignet. Viele Daten sind im Internet frei zugänglich. Du kannst sie herunter laden und bei dir in eine eigene Datenbank übernehmen.

ZUM SELBST LÖSEN

1a. Schreibe die Namen aller Länder aus, die in der Klimadatenbank aufgenommen sind.

1b. Wähle ein Land deiner Wahl und untersuche den Temperaturverlauf zwischen 1901 und 2015.

2a. Untersuche den Temperaturanstieg in allen Ländern, indem du die Temperaturdifferenz zwischen 2015 und 1901 für jedes Land auflistest.

2b. Welches Land zeigt den höchsten Zuwachs?

9. MEHRBENUTZER-DATENBANK

DU LERNST HIER...

wie du eine Datenbank für die gemeinsame Nutzung durch mehrere Benutzer einrichten kannst und welche Vorsichtsmassnahmen du dabei beachten musst.

Bisher warst du der einzige Benutzer deiner Datenbank, die sich als Datei mit der Dateierweiterung *.db* auf deinem Computer befindet. Du konntest eine bestehende Tabelle aus der Datenbank holen, bearbeiten und wieder abspeichern. Es ist aber durchaus denkbar, dass du die Datenbank auch anderen Benutzer zur Verfügung stellen möchtest, die gleichzeitig darauf zugreifen möchten. Das kannst du beispielsweise so realisieren, dass du das Verzeichnis, in dem sich die Datenbank befindet, über ein Computernetzwerk frei gibst oder die Datenbank auf einen Cloud, beispielsweise Dropbox speicherst, und anderen Benutzern Zugriffsrecht erteilst.

Solange die einzelnen Benutzer auf die Daten nur lesend zugreifen, ergeben sich bei gleichzeitiger Nutzung keine weiteren Probleme. Können die Benutzer hingegen die Daten auch verändern, muss du Vorsichtsmassnahmen ergreifen, damit die Daten der Datenbank korrekt bleiben.

MUSTERBEISPIEL

Wiederum gehst du von der Tabelle *persons* aus, die in der Datenbank *schule.db* gespeichert ist. Mit dem gleichen Programm wie in Kapitel 3 erstellst du zu Beginn die Tabelle und speicherst sie in der Datenbank.

```
from dbtable import *

persons = DbTable('id','name','vorname','wohnort','geschlecht','jahrgang')
persons.insert(1,'Huber','Lia','Bern','w', 2002)
persons.insert(2,'Meier','Luca','Basel','m', 2003)
persons.insert(3,'Frech','Marc','Bern','m', 2000)
persons.insert(4,'Bauer','Jan','Luzern','m', 2003)
persons.insert(5,'Zwahlen','Noh','Thun','m', 2002)
persons.insert(6,'Meier','Nina','Biel','w', 2001)
print persons
persons.save('schule.db')
```

Zwei Benutzer A und B wollen nun die Datenbank gemeinsam nutzen. (Du kannst dies so simulieren, dass du zwei TigerJython-Fenster öffnest und zwei Programme so schreibst, dass sie auf die gleiche Datenbank zugreifen.)

Um die Probleme zu demonstrieren, die sich dabei ergeben können, spielst du folgendes Szenario durch: A und B führen kurz nacheinander folgendes Programm aus, um die Daten zu lesen:

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
print persons
```

Wie du erwartest, "sehen" A und B dieselben Daten. Als nächstes führt A einen Update eines

Datensatzes durch, beispielsweise ändert er den Wohnort von Bauer Paul, da dieser nach Zürich umgezogen ist.

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
print persons
persons.update(name = "Bauer", vorname = "Jan")(wohnort = "Zürich")
print persons
persons.save("schule.db")
```

Etwas später ändert B den Wohnort von Zwahlen Noe, da diese nun in Biel wohnt.

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
print persons
persons.update(name = "Zwahlen", vorname = "Noah")(wohnort = "Biel")
print persons
persons.save("schule.db")
```

Etwas später schaut sich ein Benutzer C mit folgendem Programm die Personendaten an:

```
from dbtable import *

persons = DbTable()
persons.restore("schule.db")
print persons
```

MERKE DIR...

Bei der Benutzung von Datenbanken durch mehrere Personen, welche die Datenbank verändern können, ergeben sich heikle Zugriffskonflikte. Diese müssen durch besondere Vorsichtsmassnahmen vermieden werden. Im nächsten Kapitel lernst du, wie du dabei vorgehen kannst.

ZUM SELBST LÖSEN

1. Erstelle einen Dropbox-Account und stelle die Informationen über die 100 Top Songs, die du aus der TigerJython-Datenbank extrahieren kannst, mehreren Benutzern zur Verfügung. Dazu kannst du ein lokales Dropbox-Verzeichnis synchronisieren, damit die Datenbank bei einer Veränderung automatisch in die Cloud hochgeladen wird.

2. Stelle die Song-Daten anderen Benutzern zur Verfügung, indem du die Datenbank-Datei (oder den ganzen Ordner) frei gibst. Du solltest den Nutzern aber nur Leserechte einräumen, da du der einzige sein willst, der die Daten verändern kann (d.h. der deine Songs bearbeiten kann).

3. Verändere deine Song-Daten und frage andere Nutzer, was sie zu deiner neuen Reihenfolge der besten Songs meinen. (Du kannst dieses Szenario auch allein durchspielen, indem du mehrere TigerJython-Fenster gleichzeitig öffnest.)

10. ONLINE-RESERVATIONSSYSTEM

DU LERNST HIER...

wie ein Online-Reservationssystem funktioniert. Die Bestellung von Karten für Konzerte erfolgt heute fast ausnahmslos über das Internet. Dabei wird dem Kunden gewöhnlich eine aktuelle Sitzplatzbelegung gezeigt, damit dieser per Mausklick die gewünschten Sitzplätze reservieren kann. Der Belegungsplan ist in praktisch allen Fällen in einer Datenbank auf einem Datenbankserver gespeichert und der Kunde greift über einen Web-Browser auf die Datenbank zu.

MUSTERBEISPIEL

In deinem Beispiel verwendest du zwar keinen Web-Browser, um auf einen Datenbankserver zuzugreifen, sondern ein Python-Programm, das auf eine gemeinsam zugängliche Datenbank-Datei zugreift, die beispielsweise - wie im vorhergehenden Kapitel gezeigt - auf einer Cloud (z.B. Dropbox) für mehrere Benutzer freigegeben ist. Das Prinzip bleibt aber dasselbe, denn das Einlesen der Tabelleninformation mit *restore()* und das speichern mit *save()* entspricht weitgehend einer Browserabfrage. (Du kannst das Vorgehen auch mit zwei TigerJython-Fenstern auf dem gleichen PC simulieren.)

Beim Start des Programms wird dem Benutzer eine Grafik angezeigt, aus der er in einer möglichst realistischen Sitzplatzanordnung die freien und die bereits reservierten Sitzplätze entnehmen kann. Diese Information wird aus der Datenbank bezogen. Mit einem linken Mausklick kann er nun mehrmals hintereinander einen freien Sitzplatz auswählen und dann mit einem rechten Mausklick diese Wahl bestätigen. Die Benutzer wird nun eventuell aufgefordert, sich zu identifizieren und die Reservation wird in der Datenbank vollzogen.

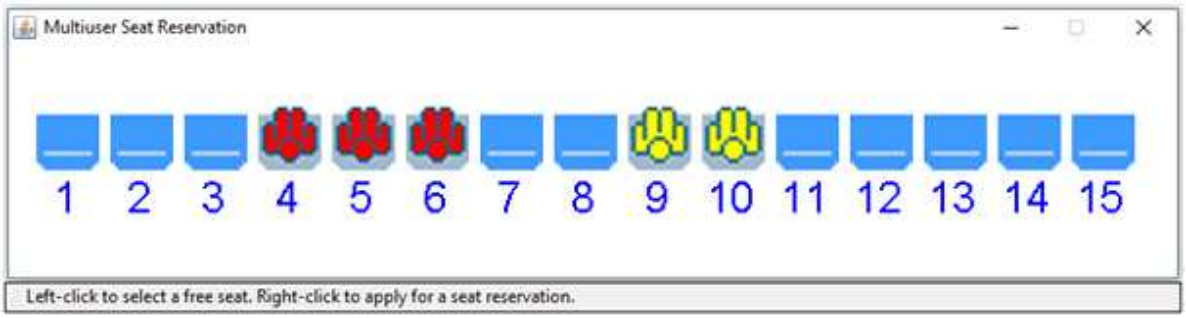
Deine Aufgabe kannst du in zwei Teile zerlegen: Das eine Teilproblem betrifft die graphische Darstellung der Sitzplätze mit der Möglichkeit, gewünschte Plätze mit einem linken Mausklick zu selektieren und mit einem rechtem Mausklick die Auswahl zu bestätigen. Die zweite Teilaufgabe besteht darin, die Datenbankmanipulationen auszuführen.

1. Teilaufgabe: Grafische Benutzeroberfläche

Für die erste Teilaufgabe verwendest du die Turtlegrafik. Um die Aufgabe zu erleichtern, sind die Sitzplätze in einer einzigen Reihe angeordnet. Trotzdem ist der Aufwand für die graphische Benutzeroberfläche recht gross und du musst deine Kenntnisse über Turtlegrafik und Mausevents heranziehen, um die Einzelheiten zu verstehen.

```
from gturtle import *
import time

def apply():
    if selected == []:
        setStatusText("Please make a seat selection.")
        return
    print "Selected seats:", selected
    for seat in selected:
        occupied.append(seat)
    renew()
    setStatusText("Your reservation is confirmed.")

def renew():
    for seat in range(1, 16):
        setPos(toSeatPos(seat))
```

```
            if seat in occupied:
                drawImage("sprites/seat_2.png")
            else:
                drawImage("sprites/seat_0.png")
        selected.clear()

def onMouseHit(x, y):
    if isRightMouseButton():
        apply()
        return
    if  y < -20 or y > 20 or x < -300 or x > 300:
        return
    seat = toSeat(x)
    setPos(toSeatPos(seat))
    if seat in occupied:
        setStatusText("This seat is occupied.")
        return
    if seat not in selected:
        drawImage("sprites/seat_1.png")
        selected.append(seat)
    else:
        drawImage("sprites/seat_0.png")
        selected.remove(seat)

def toSeatPos(seat):
    return -320 + seat * 40, 0

def toLabelPos(seat):
    return -322 + seat * 40, -40

def toSeat(x):
    return int((x + 300) / 40 + 1)

# --------------- main -----------------------------
selected = []
occupied = []
makeTurtle(mouseHit = onMouseHit)
setTitle("Multiuser Seat Reservation")
addStatusBar(20)
ht()
pu()
for seat in range(1, 16):
    setPos(toSeatPos(seat))
    drawImage("sprites/seat_0.png")
    setPos(toLabelPos(seat))
    label(str(seat), adjust = "c")

setStatusText("Left-click to select a seat. Right-click for reservation.")
while not isDisposed():
    time.sleep(1)
```



2. Teilaufgabe: Datenbankmanipulationen

Zuerst entwirfst du das Konzept der Datenbank mit der Tabellenstruktur, indem du dir überlegst, welche Informationen in welchen Tabellenfeldern abgespeichert werden. In einfachen Fällen genügt eine einzige Tabelle, in den meisten Fällen ist es aber angebracht, mehrere miteinander verknüpfte Tabellen zu verwenden. Bei deinem Reservationssystem ist es ziemlich offensichtlich, zwei Tabellen zu verwenden, eine Personentabelle *customer* mit den Kundendaten (Felder *personid, name, firstname, creditcard*) und eine Tabelle *reservation* mit der Sitzplatz-Belegung (Felder: *seatnum, booked, custid*).

reservation

seatnum	booked	custid
1	N	0
2	N	0
3	N	0
4	N	0

customer

persid	name	firsname	creditcard
1	Smith	John	3421 1002 7768 4646
2	Berger	Marie	4431 5657 4356 9856
3	Mercier	jean	4453 6547 6577 9901

booked kann die Werte 'N' (frei) und 'Y' (reserviert) annehmen. Ist ein Sitzplatz nicht reserviert, so wird als *custid* 0 eingetragen. Um die Aufgabe zu vereinfachen, betrachtest du vorerst nur die Tabelle *reservation* und erstellst und initialisierst diese mit einem einfachen Programm für die 16 Sitze in der Datenbank *casino*.

```
from dbtable import *

reservation = DbTable("seatnum", "booked", "custid")
for seat in range(1, 16):
    reservation.insert(seat, 'N', 0)
print reservation
reservation.save("casino")
```

Der Ablauf einer Reservation geht folgendermassen vor sich: Ein Benutzer A fragt nach der aktuellen Belegung der Sitzplätze. Diese Information wird aus der Datenbank geholt und grafisch darstellt. Während einer gewissen Bearbeitungszeit wählt A mit linken Mausklicks einen oder mehrere freie Plätze und bestätigt diese Wahl am Ende durch einen rechten Mausklick. Er wird nun aufgefordert, seine Personendaten einzugeben (falls er noch nicht Kunde ist) und die Reservation wird in der Datenbank vollzogen.

Bei diesem Ablauf kann sich aber folgendes Problem ergeben: Falls während der Bearbeitungszeit von A ein zweiter Benutzer B ebenfalls eine Reservation vornimmt, erhält dieser zu Beginn die gleiche aktuelle Sitzplatz-Belegung wie A und es kann durchaus sein, dass A und B dieselben Plätze reservieren möchten. Je nachdem, wer weniger lang überlegt, sagen wir A, erhält dieser die Plätze, aber B merkt davon nichts, da ja die Datenbank von sich aus keine Rückmeldungen an B machen kann. Wenn B seine Wahl bestätigt, gibt es einen schwerwiegenden **Zugriffskonflikt** für die gemeinsam gewählten Sitze.

Was ist zu tun, um den Zugriffskonflikt zu vermeiden? Eine Lösung besteht darin, dass bei der Bestätigungsoperation mit dem rechten Mausklick das Programm nachmals prüft, ob die Plätze immer noch frei sind. Ist dies nicht der Fall, so erhält der Benutzer eine Rückmeldung, dass die Plätze in der Zwischenzeit leider bereits vergeben wurden.

Im folgenden Programm ist dieser Mechanismus implementiert, was allerdings einen Zusatzaufwand mit sich bringt. Das Programm schreibt bei der Bestätigungsoperation sogar aus, welche der Plätze bereits vergeben wurden.

```
from gturtle import *
from dbtable import *
import time

# ---------------- Database management -------------
def makeReservation():
    # Get all reserved seats
```

```
    reserved = []
    reservation = DbTable()
    reservation.restore("casino")
    for row in reservation.select():
        if row[1] == 'Y':
            reserved.append(row[0])
    # Check if some selected seats are already reserved
    conflict = []
    for seat in selected:
        if seat in reserved:
            conflict.append(seat)
    if conflict != []:
        return conflict
    # No conflict-> make reservation
    for seat in selected:
        reservation.update(seatnum = seat, _booked = 'Y')
    reservation.save("casino")
    return []

# --------------- Callback executors ------------
def apply():
    if selected == []:
        setStatusText("Please make a seat selection.")
        return
    taken = makeReservation()
    if taken == []:
        setStatusText("Your reservation is confirmed.")
        renew()
    else:
        setStatusText("Seat(s) " + str(taken) +
                      " already taken. Left-click to deselect.")

def renew():
    reservation = DbTable()
    reservation.restore("casino")
    for row in reservation.select():
        if row[1] == 'Y':
            occupied.append(row[0])
    for seat in range(1, 16):
        setPos(toSeatPos(seat))
        if seat in occupied:
            drawImage("sprites/seat_2.png")
        else:
            drawImage("sprites/seat_0.png")
    selected.clear()
# -------------- Mouse callback --------------------
def onMouseHit(x, y):
    if isRightMouseButton():
        apply()
        return
    if  y < -20 or y > 20 or x < -300 or x > 300:
        return
    seat = toSeat(x)
    setPos(toSeatPos(seat))
    if seat in occupied:
        setStatusText("This seat is occupied.")
        return
    if seat not in selected:
        drawImage("sprites/seat_1.png")
        selected.append(seat)
    else:
        drawImage("sprites/seat_0.png")
        selected.remove(seat)
```

```
# ----------------- Coordinate convertions ---------
def toSeatPos(seat):
    return -320 + seat * 40, 0

def toLabelPos(seat):
    return -322 + seat * 40, -40

def toSeat(x):
    return int((x + 300) / 40 + 1)

# --------------- main -----------------------------
selected = []
occupied = []
makeTurtle(mouseHit = onMouseHit)
setTitle("Multiuser Seat Reservation")
addStatusBar(20)
ht()
pu()
for seat in range(1, 16):
    setPos(toSeatPos(seat))
    drawImage("sprites/seat_0.png")
    setPos(toLabelPos(seat))
    label(str(seat), adjust = "c")

renew()
setStatusText("Left-click to select a seat. Right-click for reservation.")
while not isDisposed():
    time.sleep(1)
```

MERKE DIR...

Der Zugriffskonflikt bei Mehrbenutzer-Datenbanken kann so gelöst werden, das vor dem Update der Datenbank nochmals geprüft wird, ob in der Zwischenzeit ein anderer Benutzer die Daten verändert hat.

ZUM SELBST LÖSEN

1. Erweitere das Reservationssystem auf eine zweidimensionale Saalanordnung.

11. BILDDATENBANK

DU LERNST HIER...

wie du eine Datenbank dafür verwenden kannst, digitalisierte Bilder in Tabellenfeldern abzuspeichern. So wie Text, kannst du auch die Pixel eines Bildes in Bytes codieren, beispielsweise den Rot-, Grün- und Blauanteil in drei Bytes R,G,B, die du als Zahlen 0..255 auffassen kannst.

MUSTERBEISPIEL

Du willst zu jeder Person einer Personendatenbank auch noch ihr Foto abspeichern. Wiederum gehst du von der Tabelle *persons* aus, so wie du sie bereits im Kapitel 3 verwendet hast, fügst aber ein zusätzliches Feld "*image*" hinzu. Die Personenbilder im Bildformat GIF, PNG oder JPG müssen sich im gleichen Verzeichnis wie das Programm befinden. Hast du gerade keine Bilder zur Verfügung, so kannst du ein kleine Bildsammlung mit 24 Cartoons von *http://www.tigerjython4kids.ch/download/cartoons.zip* herunterladen und auspacken. Die Funktion *getBytes(filename)* liest die binären Daten aus der angegebenen Bilddatei und stellt sie als Bytebuffer zur Verfügung. Sie können genau gleich wie andere Felder bei *insert()* angegeben werden.

```
from dbtable import *

lia = getBytes("lia.png")
luca = getBytes("luca.png")
tim = getBytes("tim.png")
jan = getBytes("jan.png")
noah = getBytes("noah.png")
nina = getBytes("nina.png")

persons = DbTable('id', 'name', 'vorname', 'wohnort',
                  'geschlecht', 'jahrgang', 'image')
persons.insert(1,'Huber','Lia','Bern','w', 2002, lia)
persons.insert(2,'Meier','Luca','Basel','m', 2003, luca)
persons.insert(3,'Frech','Tim','Bern','m', 2000, tim)
persons.insert(4,'Bauer','Jan','Luzern','m', 2003, jan)
persons.insert(5,'Zwahlen','Noah','Thun','m', 2002, noah)
persons.insert(6,'Meier','Nina','Biel','w', 2001, nina)
print persons
persons.save('schule.db')
```

Beim Ausschreiben der Tabelle mit *print* "siehst" du die Bilder allerdings nur als Bytewerte. Um sie als Bild anzuzeigen, benötigst du ein Grafikfenster (hier ein Turtlefenster) und durchläufst in einer for-Schleife die einzelnen Zeilen.

```
from dbtable import *
from gturtle import *

makeTurtle()
ht()
pu()
persons = DbTable()
persons.restore('schule.db')
x = -320
for record in persons:
    setPos(x, 0)
```

```
    drawImage(record.image)
    setPos(x, -100)
    label(record.vorname + " " + record.name, adjust = 'c')
    x += 130
```

Ergebnis:

MERKE DIR...

Um Bilder in eine Tabelle einzufügen, musst du diese mit *getBytes()* als Bytebuffer einlesen und sie in *insert()* angeben. Nachher kannst du alle Tabellenoperationen wie üblich anwenden.

ZUM SELBST LÖSEN

1. Erstelle eine Personendatenbank mit eigenen Bildern.

2. Schreibe ein Programm, das wiederholt in einem Eingabedialog einen Vornamen abfragt und dann den Vornamen, Namen, Wohnort und Foto im Turtlefenster anzeigt.

3a. Im Folgenden verwendest du die Tabelle *countries* von der Datenbank *tigerjython.db* aus der Distribution von TigerJython. Schreib diese Tabelle im Ausgabefenster aus und schau dir die Feldnamen und die Daten deines Heimatlandes an. Die siehst hier auch den zweistelligen *country_code* aller Länder.

3b. Die Tabelle *flags* aus der Datenbank *tigerjython.db* enthält den *country_code* und die Landesflagge im GIF-Format. Schreibe ein Programm, das wiederholt als Eingabe den *country_code* verlangt und dann in einem Turtlefenster die Flagge anzeigt.

3c. Wie 3b, aber es wird noch der Name des Landes unterhalb der Flagge ausgeschrieben.

3d. Wie 3c, aber fang den Fehler ab, falls man einen falschen *country_code* eingibt.

4. Erstelle ein Quiz wie folgt: Es erscheint eine zufällige Landesflagge und darunter werden die Namen von drei Länder in der Reihenfolge 1, 2, 3 ausgeschrieben.

   Eines der Länder entspricht der gezeigten Flagge. In einem Eingabe-Fenster wird der Spieler aufgefordert, das Land der Flagge anzugeben (als Zahl 1, 2, 3). Das richtige Land wird danach angezeigt. Man kann solange spielen, bis man im Eingabedialog "Abbrechen" wählt.

ANHANG : SOUNDCLIPS

Um den Soundclip der Datei *frog.wav* abzuspielen, verwendet man das Modul *soundsystem*, das hier dokumentiert ist.

```
from soundsystem import *
openSoundPlayer("frog.wav")
play()
```

Die Daten einer Sounddatei können wie jede andere binäre Datei in einer Tabelle gespeichert werden. Dazu liest man die Datei mit *getBytes()* in einen Bytebuffer ein und fügt diesen mit *insert()* in die Tabelle. Nachher wird die Tabelle mit *save()* in der Datenbank gespeichert.

```
from dbtable import *
frogSound = getBytes("frog.wav")
animals = DbTable('name', 'sound')
animals.insert("frog", frogSound)
animals.save("animalsound.db")
```

Mit *restore()* liest man die Daten aus der Datenbank aus und erhält dabei einen Bytebuffer. Diesen kann man mit *storeBytes()* wieder in eine Datei schreiben und hat damit die Originaldatei bitgenau wieder hergestellt.

```
from dbtable import *
animals = DbTable()
animals.restore('animalsound.db')
resultSet = select(name = "frog")
storeBytes(resultSet[0][1], "myfrog.wav")
```

Das Verfahren funktioniert genau gleich für jede beliebige Datei, also auch für jedes andere Audioformat, z.B. MP3. In der Datenbank gespeicherte WAV-Dateien (nur solche) kann man aber auch direkt abspielen, ohne sie wieder in eine Datei zu schreiben:

```
from dbtable import *
from soundsystem import *
animals = DbTable()
animals.restore('animalsound.db')
resultSet = animals.select(name = "frog")
openSoundPlayer(resultSet[0][1])
play()
```

In der Datenbank *tigerjython.db* der Distribution von TigerJython befindet sich die Tabelle *birdsounds* mit den Namen und Zwitschertönen einiger Singvögel. Die Clips können wie folgt nacheinander ausgeschrieben und abgespielt werden:

```
from dbtable import *
from soundsystem import *
import time
birdsounds = DbTable()
birdsounds.restoreFromTJ("tigerjython.db")
for bird in birdsounds:
    print bird.name
    openSoundPlayer(bird.sound)
    blockingPlay()
    time.sleep(3)
```

Dokumentation Datenbanken

Module imports: **from dbtable import***

Abstraktion von Datenbank-Tabellen ohne SQL

Klasse Table:

Befehl	Aktion
tbl = Table(fieldname1, fieldname2,...)	erzeugt ein Tabellenobjekt mit beliebig vielen Feldnamen (als String). Die Feldnamen können in ein Tupel gepackt sein
tbl = Table(anotherTable)	erzeugt einen (tiefen) Klone des Tabellenobjekts
tbl = Table()	erzeugt eine leere Tabelle, die für restore(), restoreFromTJ() oder importFromCSV() verwendet werden kann
tbl.insert(value1, value2,...)	fügt eine Zeile mit den gegebenen Werten in die Tabelle. Die Werte können in ein Tupel gepackt sein. Erlaubte Datentypen: str, int, float, BLOB (binär). Es müssen die Werte aller Felder angegeben werden. Aus der zuerst eingegenen Zeile wird der Datentyp der Felder bestimmt. Binäre Felderwerte sind Bytearrays (Rückgabewert von getBytes(filename))
tbl.insertMany(liste)	fügt mehrere Zeilen, die in liste (oder tuple) enthalten sind (beispielsweise von einer select() Rückgabe). Aus der zuerst eingegenen Zeile wird der Datentyp der Felder bestimmt (int, float oder str)
for row in tbl: do_something_ with row.fieldname	Durchlaufen aller Zeilen und Verwenden der Feldwerte
tbl.select(fieldname1, fieldname2, ..., pattern)	gibt ein Tupel aller Zeilen mit den angegebenen Feldern zurück. Wird mit sort() ein Sortierattribut angegeben, so werden die Zeilen in der Sortierreihenfolge zurückgegeben. pattern ist eine Sequenz mit Gleichheitsbedingungen attribute = wert. Falls ein Attribut fehlt, erfüllen alle Werte die Bedingung. Beispiel: select("name", "vorname", "alter", name = "Meyer", vorname = "Hans") gibt die Felder "name", "vorname" und "alter" aller "Meyer Hans" zurück. Die Werte haben den Datentyp str, int oder float
tbl.sort(fieldname)	definiert einen Feldnamen (String) als Sortierattribut. Tabellenansichten und select() werden nachher in aufsteigender Reihenfolge dieses Attributs ausgegeben. Ohne Angabe eines Sortierattributs, ist die Reihenfolge der Zeilen undefiniert (aber normalerweise in der Reihenfolge der mit insert() eingefügten Zeilen)
tbl.sort(fieldname, False)	dasselbe, aber in absteigender Reihenfolge
tbl.getAttributes()	gibt ein Tupel mit den Feldnamen (Attributen) zurück
tbl.view()	zeigt die Tabellenwerte formatiert in der Konsole (sortiert gemäss einem Sortierattribut)
tbl.view(fieldname1, fieldname2, ..., pattern)	dasselbe, aber es werden nur die angegebenen Felder von Zeilen angezeigt, welche die Gleichheitsbedingungen erfüllen (wie select())
tbl.getView()	dasselbe, aber Rückgabe als formatieren String

tbl.getView(fieldname1, fieldname2, ..., pattern)	dasselbe, aber es werden nur die angegebenen Felder von Zeilen zurück gegeben, welche die Gleichheitsbedingungen erfüllen (wie select())
print tbl	dasselbe wie tbl.view()
len(tbl)	gibt die Anzahl Tabellenzeilen zurück
tbl.delete(pattern)	löscht alle Zeilen, welche die Gleichheitsbedingungen in pattern erfüllen (wie bei select()). Beispiel: delete(name = "Meyer", vorname = "Hans" löscht alle Zeilen mit Meyer Hans
tbl.update(pattern)(fieldname1 = value1, fieldname2 = value2,...)	ersetzt bestimmte Tabellenwerte. Es werden dazu zwei Parameterklammern verwendet. In der ersten Klammer werden mit pattern wie in select() die Zeilen ausgewählt. In der zweiten Parameterklammer werden die neuen Feldwerte der ausgewählten Zeilen angegeben. Beispiel: update(name = "Meyer", vorname = "Hans")(wohnort = "Basel") setzt den Wohnort aller "Meyer Hans" auf "Basel". Ist die erste Parameterklammer leer, so werden alle Zeilen modifiziert
tbl.join(otherTable, left, right)	gibt eine Tabelle zurück, die einer Tabellenverbindung (join) der vorhandenen Tabelle mit otherTable entspricht, wobei die attribute left und right übereinstimmen müssen. Die Vergleichsattribute werden nicht übernommen. Beispiel: person.join(sport, person.id = sport.id)
tbl.join(otherTable, left, right, True)	dasselbe, aber es werden auch die Vergleichsattribute übernommen (diese müssen unterschiedliche Namen haben)
tbl.join(otherTable)	dasselbe, aber es werden alle Zeilen beider Tabellen miteinander verbunden (Kreuzprodukt)
tbl.saveTable(databaseName, tableName)	speichert die Werte aus table in einer SQL-Datenbanktabelle einer SQLite-Datenbank mit gegebenen Dateinamen unter dem gegebenen SQLite-Datenbank-Tabellennamen. Falls die Datenbank nicht existiert, wird sie erzeugt. Eine bestehende Tabelle mit gleichem Namen wir gelöscht. Die Feldnamen (Attribute) und Datentypen werden übernommen. Wird tableName nicht angegeben, so wird der Variablenname als SQLite-Tabellenname verwendet
tbl.restoreTable(databaseName, tableName)	holt die mit saveTable() in der SQLite-Datenbank gespeicherten Werte aus der gegebenen SQLite-Datenbank-Tabelle und gibt ein Table-Objekt mit den wiederhergestellten Feldnamen (Attributen) zurück. Wird tableName nicht angegeben, so wird der Variablen als SQLite-Tabellenname verwendet.
tbl.restoreFromTJ(databaseName, tableName)	dasselbe, aber die Datenbank wird zuerst aus der TigerJython-Distribution (tigerjython2.jar) geholt und in das Verzeichnis des Programms kopiert
tbl.importFromCSV(filename, delimiter)	importiert den Inhalt einer CSV-Datei (Textdatei). Die Felder müssen durch das Trennzeichen delimiter getrennt sein. Jede Zeile muss genau die gleiche Anzahl Felder haben. Aus der ersten Zeile wird der Typ der Felder (int, float, str) ermittelt
tbl.exportToCSV(filename, delimiter, fieldname1, fieldname2, ..., pattern)	exportiert die Tabellendaten in eine CSV-Datei, wobei die Felder mit dem angegebenen Trennzeichen getrennt sind. Es werden nur die angegebenen Felder übernommen. pattern ist eine Sequenz mit Gleichheitsbedingungen attribute = wert. Fehlen fieldname und pattern, so wird die ganze Tabelle übernommen (wie bei select())

getBytes(filename)	gibt einen Bytearray (Typ array.arrray) der Datei filename zurück (absoluter oder relativer Pfad bezüglich des Verzeichnis, in dem sich tigerjython2.jar befindet)
storeBytes(buffer, filename)	schreibt den Bytearray (Typ array.arrray) in die Datei filename (ersetzt vorhandene Datei)
showDbInfo(database)	schreibt von einer Datenbank alle Tabellennamen und ihre Feldnamen aus
showDbInfoTJ(database)	schreibt von einer Datenbank in der TigerJython-Distribution alle Tabellennamen und ihre Feldnamen (Attribute) aus

Tabellen der Datenbank *tigerjython.db* (in tigerjython2.jar integriert)

restoreFromTJ("tigerjython.db")

Tabelle	Feldnamen	Anzahl Datensätze
babynames	rank, forename	100
boynames	rank, forename	50
girlnames	rank, forename	50
firstnames	fname, fwoman, fman	54'379
flags	country_code, flag	248
countries	country_en, country_de, country_code, continent, capital, ...	249
capitals	city_en, city_de, country_code, population...	240
meantemp	country, 1900, 1901, 1902, ...2015	89
mountains	name, county, hight	28
topsongs	title, artist, rank	100

ARBEITSBLATT 4: DATENBANKEN

DATENBANKEN ÜBERALL

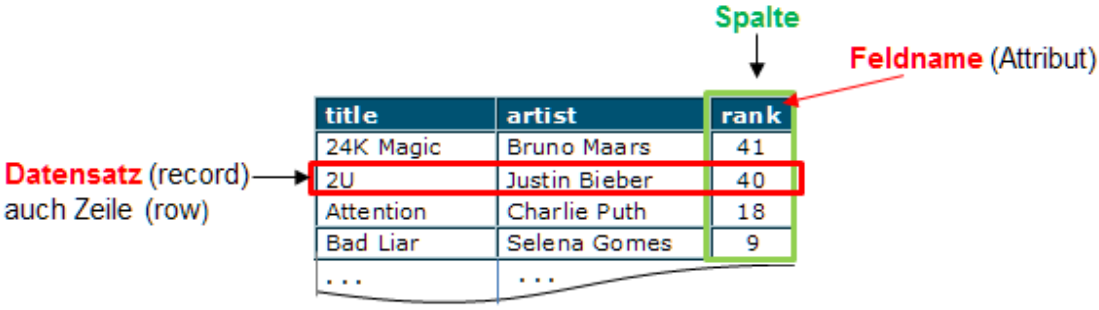
In unserer digitalen Gesellschaft hinterlässt du täglich unzählige Datenspuren, beispielsweise auf Überwachungskameras, beim Vorweisen von Abonnementen und Eintrittskarten, beim Zahlen mit Kreditkarten, beim Telefonieren und Versenden von SMS und natürlich, wenn du dich in sozialen Medien bewegst. Die enorme Menge von Informationen aller erfassten Personen werden digitalisiert und so in Datenbanken abgelegt, dass mehrere Nutzer von verschiedenen Orten mit beliebigen Systemen darauf zugreifen können.

FRAGE 1: Warum ist Erfassung all dieser Daten ohne Verwendung des Computers unmöglich?

FRAGE 2: Welche Auswirkungen kann die Erfassung von Personendaten auf dich haben?

TABELLEN VERWENDEN

Informationen werden als Daten in einer Datenbank oft in Tabellenform abgespeichert. Du kennst Tabellen aus dem täglichen Leben und weisst, dass Tabellen in Zeilen und Spalten strukturiert sind.



Für dieses Arbeitsblatt verwendest du eine Musikchart (Hitliste) mit den hundert aktuell beliebtesten Songs. In einer Tabelle werden der Titel, der Name des Sängers und die Platzierung aufgeführt.

AUFGABE 1:

Merke dir die oben notierten Begriffe. Das TigerJython-Modul *DbTable* enthält Befehle, die es ermöglichen, auf eine einfache Art mit Datenbanktabellen umzugehen.

Um diese Befehle verwenden zu können, musst du das Modul *dbtable* importieren:

```
from dbtable import *
```

Damit du die Daten nicht selbst eintippen musst, kannst du zuerst mit

```
topsongs = DbTable()
```

ein leeres Tabellenobjekt erzeugen und dann mit dem Befehl *restoreFromTJ()* die Daten aus der Distribution von TigerJython holen.

```
topsongs.restoreFromTJ("tigerjython.db")
```

Schliesslich stellst du die Tabelle im Ausgabefenster mit dem *view*-Befehl dar:

```
topsongs.view()
```

Zusammengefasst:

```
from dbtable import *

topsongs = DbTable()
topsongs.restoreFromTJ("tigerjython.db")
topsongs.view()
```

AUFGABE 2

Führe das Programm aus und betrachte die Tabelle im Ausgabefenster.

FRAGE 3: Nach welcher Spalte ist die Tabelle sortiert?

FRAGE 4: Welches sind die fünf beliebtesten Songs?

FRAGE 5: Wie viele Songs werden von Justin Bieber gesungen?

Um die Fragen 4 und 5 zu beantworten, musst du ziemlich viel "Knochenarbeit" leisten. Stell dir vor, wie es wäre, wenn die Tabelle 100'000 Zeilen hätte! In deinem Programm kannst du den Computer mit einem einzigen Befehl anweisen, die Tabelle zu sortieren. Willst du sie beispielsweise in der Reihenfolge der Spalte *rank* sortieren lassen, so schreibst du

```
topsongs.sort("rank")
```

Du kannst auch verlangen, dass nur Zeilen ausgeschrieben werden, die eine bestimmte Bedingung erfüllen, beispielsweise dass der Sänger *Justin Bieber* heisst. Du schreibst dazu

```
topsongs.view(artist = "Justin Bieber")
```

AUFGABE 3

Löse Fragen 4 und 5 mit einem Programm.

TABELLEN SPEICHERN UND WIEDER HOLEN

Tabellenvariablen verhalten sich gleich wie andere Programmvariable. Wie du weisst, gehen Variablenwert bei Programmende verloren. Damit du die Tabellendaten in einem anderen Programmlauf wieder verwenden kannst, musst du daher die Tabelle in einer Datenbankdatei speichern. Die Tabelle *topsongs* kannst du beispielsweise in der Datenbankdatei *mediathek.db* wie folgt speichern:

```
topsongs.save("mediathek.db")
```

Statt *Datenbankdatei* sagt man auch nur kurz *Datenbank* und du findest diese nach dem *save*-Befehl im gleichen Verzeichnis wie dein Programm.

Um die Tabelle in einem anderen Programm wieder zu verwenden, holst du ihre Werte mit:

```
topsongs = DbTable()
topsongs.restore("mediathek.db")
```

AUFGABE 4

Speichere deine nach Rang sortierte Tabelle *topsongs* in der Datenbank *mediathek.db*

AUFGABE 5

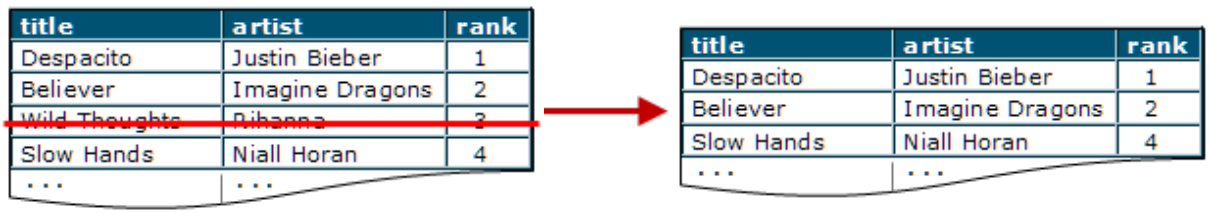
Hole die Tabelle *topsongs* aus der Datenbank *mediathek.db* zurück und schreibe sie im Ausgabefenster aus. Beachte, dass die Tabelle jetzt nach Rang sortiert ist.

DATENSÄTZE LÖSCHEN

Um einen Datensatz zu löschen, verwendest du den *delete*-Befehl, musst aber mit einer Bedingung angeben, welchen Datensatz du löschen willst Die Bedingung bezieht sich auf einen Wert eines bestimmten Feldes, beispielsweise löschst du mit

```
topsongs.delete(title = "Wild Thoughts")
```

die Datensätze, welche den Titel "Wild Thoughts" haben. Trifft dies für mehrere Datensätze zu, so werden alle gelöscht.



Um den Datensatz genauer festzulegen, kannst du auch kommagetrennt mehrere Bedingungen angeben, die alle erfüllt sein müssen.

AUFGABE 6

Hole die Tabelle *topsongs* aus der Datenbank *mediathek.db* zurück und lösche alle Songs, die von *Justin Bieber* gesungen werden. Zeige die Tabelle an, aber speichere sie nicht, damit du immer noch die unveränderte Datenbank verwenden kannst.

DATENSÄTZE EINFÜGEN

Um einen neuen Datensatz einzufügen, verwendest du den *insert*-Befehl, z.B. fügst du einen Song von *Mani Matter* auf Platz 101 so ein:

```
topsongs.insert("Bim Coiffeur", "Mani Matter", 101)
```

Beachte, dass du im Befehl die Werte aller Felder angeben musst.

title	artist	rank
. . .	. . .	. . .
Something good	Brett Eldredge	97
If I Told You	Darius Rucker	98
More Girls Like	Kip Moore	99
Its Everyday	Jake Paul	100

title	artist	rank
. . .	. . .	. . .
Something good	Brett Eldredge	97
If I Told You	Darius Rucker	98
More Girls Like	Kip Moore	99
Its Everyday	Jake Paul	100
Bim Coiffeur	Mani Matter	101

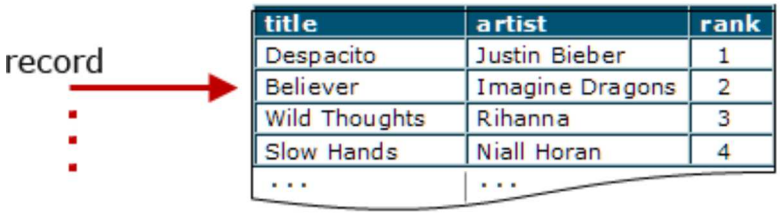
AUFGABE 7

Füge einige deiner beliebtesten Songs ein. Achte darauf, dass die Rangliste immer noch richtig bleibt, dass also nicht zwei Songs denselben Rang belegen. Du kannst auch bestimmte Songs löschen. Speichere die Tabelle zuletzt in der Datenbank *mediathek1.db* ab.

DATENSÄTZE DURCHLAUFEN

In Datenbanktabellen bilden die einzelnen Zeilen, also die Datensätze, eine Einheit. Du greifst auf Tabelleninformationen zu, indem du Datensatz um Datensatz liest. Man sagt auch, *"man durchlaufe"* dabei die Tabelle. Dazu wird am einfachsten eine *for*-Schleife verwendet. Beispielsweise kannst du die Tabelle *topsongs* aus der Datenbank *mediathek.db* mit folgendem Programm durchlaufen und die Felder *rank* und *title* ausschreiben. Dabei verwendest du den Punktoperator, um auf die Feldwerte zuzugreifen.

```
from dbtable import *

topsongs = DbTable()
topsongs.restore("mediathek.db")
for record in topsongs:
    print record.rank, record.title
```



title	artist	rank
Despacito	Justin Bieber	1
Believer	Imagine Dragons	2
Wild Thoughts	Rihanna	3
Slow Hands	Niall Horan	4
. . .	. . .	

AUFGABE 8

Sortiere die Tabelle nach Rang und schreibe dann Titel und Sänger zeilenweise aus. Beim Durchlaufen mit der *for*-Schleife kannst du mit *if*-Anweisungen beliebige Bedingungen ausprogrammieren. Sollen z.B. nur die Songs von *Justin Bieber* angezeigt werden, so schreibst du:

```
if record.artist == "Justin Bieber":
    print record.rank, record.title
```

AUFGABE 9

Schreibe die Felder *title*, *artist* für die ersten 10 Songs der Hitliste aus. Natürlich kannst du diese Aufgaben auch mit deiner eigenen Datenbank *mediathek1.db* lösen.

PERSONENDATENBANK MIT FOTOS

Um eine eigene neue Datenbank *persons* mit den Feldern *name, vorname, alter, foto* anzulegen, die den Familiennamen, Vornamen, das Alter und ein Bild der Person enthält, erstellst du zuerst eine Tabelle mit den gewünschten Feldern:

```
persons = DbTable("name", "vorname", "alter", "image")
```

Nun kannst du einzelne Personen in die Tabelle einfügen, benötigst aber von allen ein Bild (im Format GIF, PNG oder JPG). Die Bilddatei befindet sich im gleichen Verzeichnis wie dein Programm (beispielsweise für die vierzehnjährige Kunz Anna als *anna.png*). Das Bild musst du zuerst mit *getBytes()* von der Datei in eine Variable einlesen und kannst es dann in den Datensatz einfügen.

```
annaPict = getBytes("anna.png")
persons.insert("Kunz", "Anna", 14, annaPict)
```

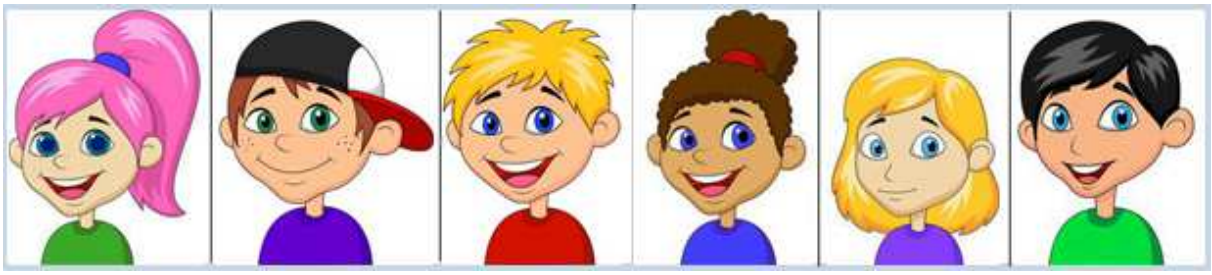
Anmerkung: Falls du gerade keine Bilder hast, so kannst du einige Comics-Bilder von http://www.tigerjython4kids.ch/download/comics.zip downloaden und im Verzeichnis, in welchen dein Programm gespeichert ist, auspacken. Du musst die Bilder eventuell umbenennen.

AUFGABE 10

Füge einige Datensätze von dir und deinen Kameraden ein und speichere die Tabelle in der Datenbank *kameraden.db.*

Da das Foto in digitalisierter Form in der Datenbank gespeichert ist, wird es beim Ausschreiben der Tabelle mit *view()* nicht als Foto dargestellt. Um das Bild zu sehen, benötigst du ein Grafikfenster, am einfachsten ein dir bekanntes Turtlefenster. Mit dem Turtlebefehl *drawImage()* kannst du an der aktuellen Position der Turtle das Bild erscheinen lassen. Als Vorlage verwendest du folgendes Programm:

```
from dbtable import *
from gturtle import *

makeTurtle()
ht()
persons = DbTable()
persons.restore('kameraden.db')
x = -320
for record in persons:
    setPos(x, 0)
    drawImage(record.image)
    x += 130
```



AUFGABE 11

Zeige die Personenbilder aus deiner Datenbank an. Schreibe dann unter jedem Bild auch noch Vornamen und Namen aus (mit dem Turtlebefehl *label()*).

AUFGABE 12

Schreibe ein Programm, dass in einem Eingabedialog nach einem Vornamen fragt und dann die betreffende Person anzeigt.

AUFGABE 13

Erweitere die Personendatenbank und das Abfragen und Anzeigen von Personen nach deinen eigenen Vorstellungen.

PERSONENDATENBANK ANDEREN ZUR VERFÜGUNG STELLEN

Um die Datenbank mehreren Personen zur Verfügung zu stellen, kannst du die Datei *kameraden.db* auf einer Cloud speichern (beispielsweise auf Dropbox oder iClound) und anderen Kameraden frei geben. Wenn du dein lokales Verzeichnis mit der Cloud automatisch synchronisierst, so werden jedem User die von dir modifizierten Daten innert wenigen Sekunden zur Verfügung stehen.

AUFGABE 14

Erstelle ein Dropbox-Konto und synchronisiere deine Datenbank. Stelle die Daten anderen Benutzern über das Internet zur Verfügung.

FRAGE 6: Welche Schwierigkeiten können sich ergeben, wenn andere Nutzer die Datenbank auch verändern können?

ANHANG

ÜBER DIE AUTOREN

Jarka Arnold war als Dozentin an der Pädagogischen Hochschule Bern für die Informatikausbildung angehender Lehrkräfte für die Sekundarstufe 1 tätig. Sie hat dabei Informatikgrundkonzepte und das Programmieren mit Java, PHP und Python vermittelt. Ihre langjährige Erfahrung in der Aus-und Weiterbildung von Informatiklehrpersonen und viele Musterbeispiele sind in diesen Lehrgang eingeflossen. Sie ist zudem verantwortlich für den Webauftritt dieses Lehrgangs.

Aegidius Plüss war an der Universität Bern Professor für Informatik und deren Didaktik und hat in dieser Tätigkeit viele Informatiklehrkräfte aus- und weitergebildet, die heute aktiv an den Schulen tätig sind. Er gilt als Urgestein in der Informatikausbildung und hat eine grosse Erfahrung mit vielen Programmiersprachen und Computersystemen. Er ist für die textliche Formulierung dieses Lehrgangs und für die didaktischen Libraries in TigerJython verantwortlich.

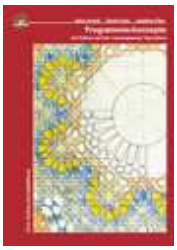
KONTAKT

Die Entwicklergruppe von TigerJython4Kids ist dankbar für jede Art von Rückmeldungen, insbesondere für Fehlermeldungen und Richtigstellungen, Anregungen und Kritik. Wir bieten auch Hilfe und Beratung bei fachlichen oder didaktischen Fragen zu Python und den in TigerJython integrierten Libraries, sowie zur Robotik-Hardware.

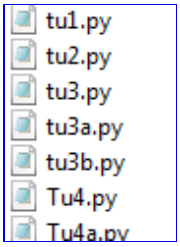
Schreiben Sie ein Email an:

help@tigerjython.com

LINKS



J. Arnold, T. Kohn, A. Plüss: Programmierkonzepte mit Python
Umfangreiches Online-Lehrmittel, mit vielen Beispielen aus verschiedenen Gebieten, geeignet für den Einsatz in weiterführenden Schulen und zum Selbststudium.
www.tigerjython.ch



Programmbeispiele
Sourcecodes aller Programmbeispiele aus *TigerJython4Kids*.
http://www.tigerjython4kids.ch/download/examples.zip

Jarka Arnold: Turtlegrafik, Robotik und Spiele mit Python
Online-Lernprogramm mit vielen lauffähigen Programmbeispielen und Aufgaben für den Einsatz im Unterricht auf der Sekundarstufe 1 und 2.
http://www.jython.ch

Tobias Kohn: Python
Eine Einführung in die Computer-Programmierung.
Ein Skript im PDF-Format.
http://jython.tobiaskohn.ch/PythonScript.pdf (166 Seiten)

Jarka Arnold, Aegidius Plüss: Python exemplarisch
Ergänzende Python-Tutorials: Visualisierung, Raspberry PI und Data Mining
http://www.python-exemplarisch.ch