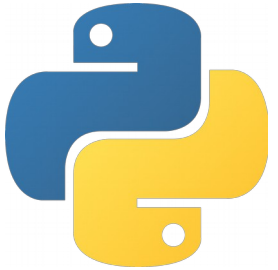
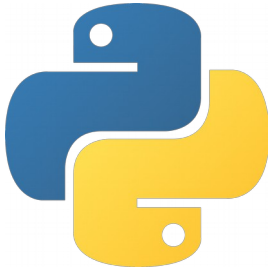
Python Datenspeicherung



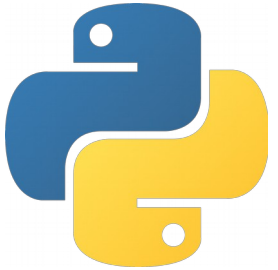
- Komprimierte Dateien lesen und schreiben-gzip
  - Mit dem Modul gzip der Standardbibliothek können Sie auf einfache Weise Dateien verarbeiten, die mit der zlib-Bibliothek erstellt wurden
  - Das Modul stellt eine Funktion namens open bereit, die sich in ihrer Verwendung an die Built-in Function open anlehnt
  - gzip.open(filename[, mode[, compresslevel])
  - Die Funktion gzip.open gibt ein Objekt zurück, das wie ein ganz normales Dateiobjekt verwendet werden kann
  - Die Parameter filename und mode sind gleichbedeutend mit denen der Built-in Function open

Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Mit dem letzten Parameter, compresslevel, können Sie angeben, wie stark die Daten beim Schreiben in die Datei komprimiert werden sollen
  - Erlaubt sind Ganzzahlen von 0 bis 9, wobei 0 für die schlechteste und 9 für die beste Kompressionsstufe steht
  - Je höher die Kompressionsstufe ist, desto mehr Rechenzeit ist auch für das Komprimieren der Daten erforderlich
  - Wird der Parameter compresslevel nicht angegeben, verwendet gzip standardmäßig die beste Kompression

Python Datenspeicherung

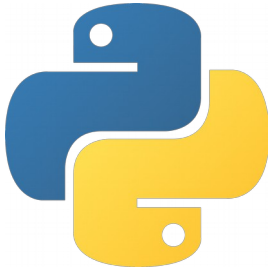


- Komprimierte Dateien lesen und schreiben-gzip
  - Beispiel

```
>>> import gzip
>>> f = gzip.open("testdatei.gz", "wb")
>>> f.write(b"Hallo Welt")
>>> f.close()
>>> g = gzip.open("testdatei.gz")
>>> g.read()
b'Hallo Welt'
```

  - In dem Beispiel schreiben wir einen einfachen bytes-String in die Datei testdatei.gz und lesen ihn anschließend wieder aus

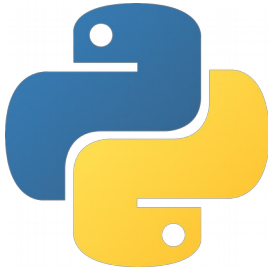
Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Weiter Module

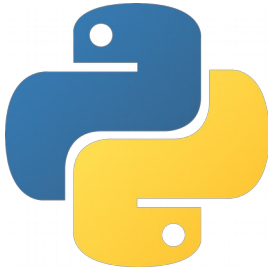
Modul	Beschreibung
zlib	Eine Low-Level-Bibliothek, die direkten Zugriff auf die Funktionen der *zlib* ermöglicht. Mit ihr ist es unter anderem möglich, Strings zu komprimieren oder zu entpacken. Das Modul `gzip` greift intern auf das Modul `zlib` zurück.
gzip	Bietet komfortablen Zugriff auf Daten, die mit der *zlib* komprimiert wurden.
bz2	Bietet komfortablen Zugriff auf Daten, die mit dem *bzip2*-Algorithmus komprimiert wurden, und ermöglicht es, neue komprimierte Dateien zu erzeugen. Auch `bz2` implementiert ein Dateiobjekt, das genauso zu handhaben ist wie die Objekte, die die Built-in Function `open` zurückgibt. In der Regel ist die Kompression von *bzip2* der von *zlib* in puncto Kompressionsrate überlegen.
zipfile	Ermöglicht den Zugriff auf *ZIP*-Archive, wie sie beispielsweise von dem bekannten Programm *WinZip* erstellt werden. Auch die Manipulation und Erzeugung neuer Archive ist möglich. Das Modul `zipfile` ist sehr umfangreich und mächtig und in jedem Fall einen näheren Blick wert.
tarfile	Implementiert Funktionen und Klassen, um die in der Unix-Welt weitverbreiteten tar-Archive zu lesen oder zu schreiben.

Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - XML
    - Das Modul xml der Standardbibliothek erlaubt es, XML-Dateien einzulesen und zu schreiben
    - XML ist eine standardisierte Beschreibungssprache, die es ermöglicht, komplexe, hierarchisch aufgebaute Datenstrukturen in einem lesbaren Textformat abzuspeichern
    - Da XML in vielen Bereichen von JSON abgelöst wurde, sollten Sie sich damit befassen, wenn Sie es benötigen

Python Datenspeicherung



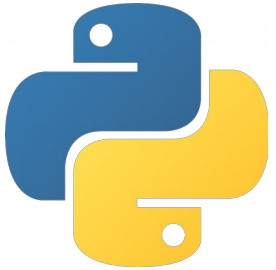
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Je mehr Daten ein Programm verwalten muss und je komplexer die Struktur dieser Daten wird, desto größer wird der Programmtechnische Aufwand für die dauerhafte Speicherung und Verwaltung der Daten

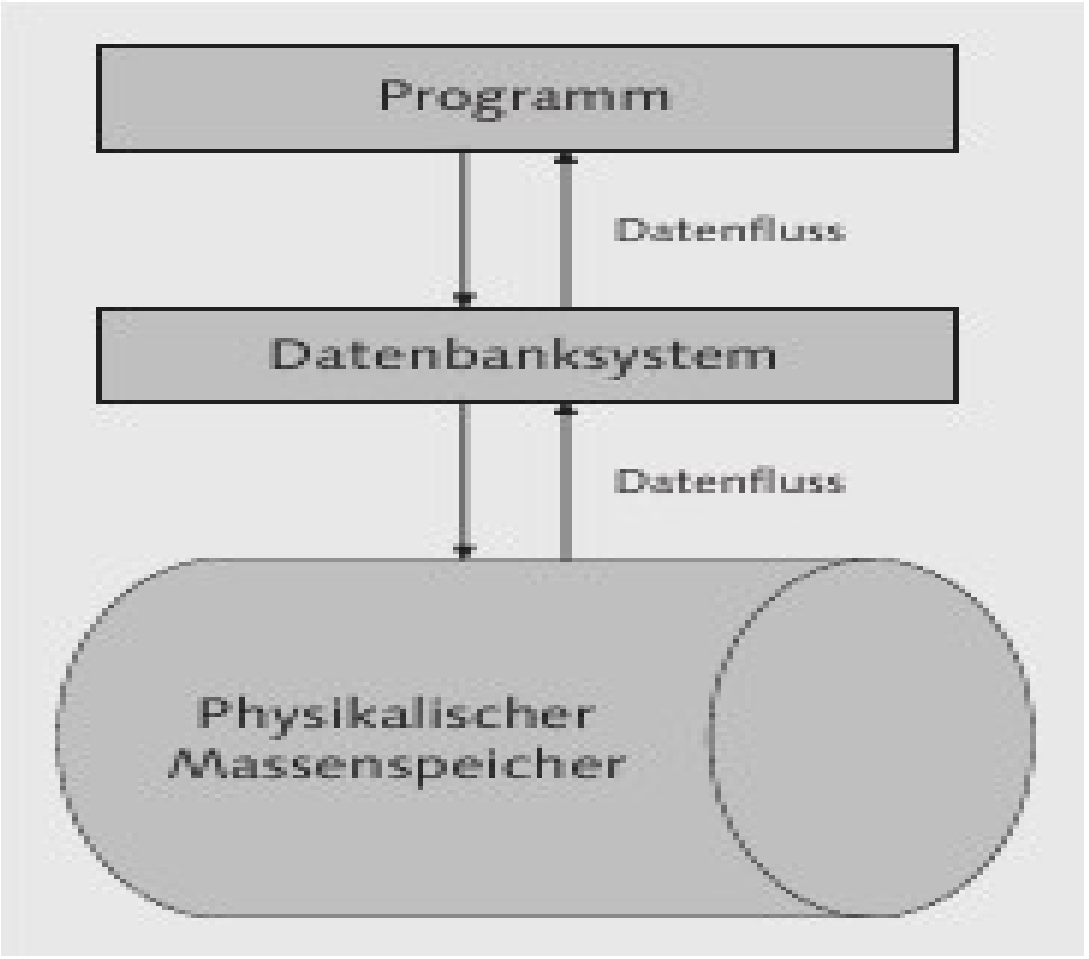
    - Außerdem müssten Aufgaben wie das Lesen, Schreiben oder Aktualisieren von Daten, die in vielen Programmen gebraucht werden, immer wieder neu implementiert werden

    - Dabei erfolgt die Kommunikation zwischen Benutzerprogramm und Datenbank über eine vereinheitlichte Schnittstelle

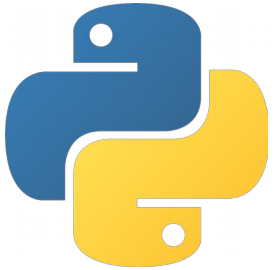
Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

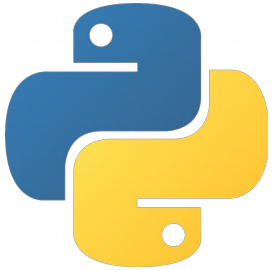


Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Verarbeitet Queries
      - Abfragen ala SQL Statements
    - Wir werden uns ausschließlich mit sogenannten relationalen Datenbanken beschäftigen, die einen Datenbestand in Tabellen organisieren
    - Für die Abfragen in relationalen Datenbanken wurde eine eigene Sprache entwickelt, deren Name SQL ist
    - Wir werden hier nur auf grundlegende SQL-Befehle eingehen, die nötig sind, um Datenbanken und deren Anwendung in Python zu verdeutlichen

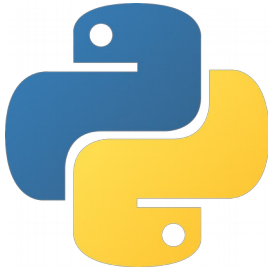
Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Hinweis
      - Neben der Abfragesprache SQL ist in Python auch die Schnittstelle der Datenbankmodule standardisiert. Dies hat für den Programmierer den angenehmen Nebeneffekt, dass sein Code mit minimalen Anpassungen auf allen Datenbanksystemen lauffähig ist, die diesen Standard implementieren.

Python Datenspeicherung

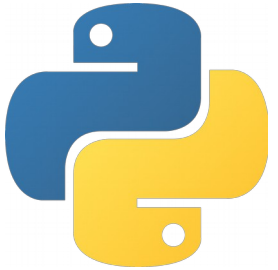


- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Beispieldatenbanktabelle Lager
      - Eine entsprechende Tabelle mit ein paar Beispieldatensätzen

Fachnummer	Seriennummer	Komponente	Lieferant	Reserviert
1	26071987	Grafikkarte Typ 1	FC	0
2	19870109	Prozessor Typ 13	LPE	57
10	06198823	Netzteil Typ 3	FC	0
25	11198703	LED-Lüfter	FC	57
26	19880105	Festplatte 128 GB	LPE	12

Python Datenspeicherung

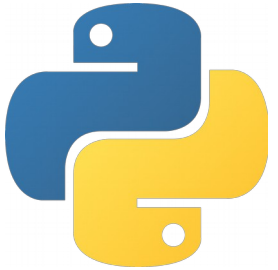


- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Beispieldatenbanktabelle Lieferanten
      - Eine entsprechende Tabelle mit ein paar Beispieldatensätzen

Kurzname	Name	Telefonnummer
FC	FiboComputing Inc.	011235813
LPE	LettgenPetersErnesti	026741337
GC	Golden Computers	016180339

Python Datenspeicherung

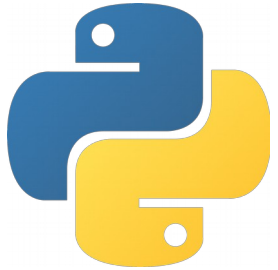


- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Beispieldatenbanktabelle Kunden
      - Eine entsprechende Tabelle mit ein paar Beispieldatensätzen

Kundennummer	Name	Anschrift
12	Heinz Elhurg	Turnhallenstr. 1, 3763 Sporthausen
57	Markus Altbert	Kämperweg 24, 2463 Duisschloss
64	Steve Apple	Podmacstr. 2, 7467 Iwarhausen

Python Datenspeicherung



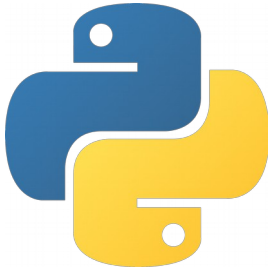
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Wir brauchen Routinen, um neue Kunden und Lieferanten hinzuzufügen, ihre Daten beispielsweise bei einem Umzug zu aktualisieren oder sie auf Wunsch aus unserer Datenbank zu entfernen

    - Auch in die Tabelle »Lager« müssen wir neue Einträge einfügen und alte löschen oder anpassen

    - Um die Datenbank aktuell zu halten, benötigen wir also Funktionen zum Hinzufügen und Löschen

Python Datenspeicherung



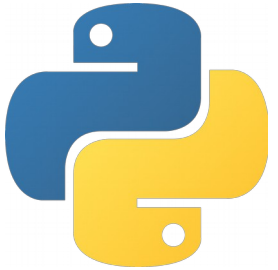
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

```
import sqlite3
connection = sqlite3.connect(“lagerverwaltung.db“)
```

  - In Python müssen Sie das Modul sqlite3 importieren, um mit der Datenbank zu arbeiten

  - Anschließend können Sie eine Verbindung zu der Datenbank aufbauen, indem Sie die connect-Funktion, die ein Connection-Objekt zu der Datenbank zurückgibt, aufrufen und ihr den Dateinamen für die Datenbank übergeben

Python Datenspeicherung



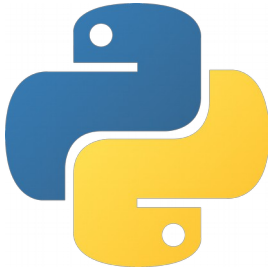
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Oft benötigt man eine Datenbank nur während des Programmlaufs, um Daten zu verwalten oder zu ordnen, ohne dass diese dauerhaft auf der Festplatte gespeichert werden müssen

    - Zu diesem Zweck gibt es die Möglichkeit, eine Datenbank im Arbeitsspeicher zu erzeugen, indem Sie statt eines Dateinamens den String ":memory:" an die connect-Methode übergeben

      connection = sqlite3.connect(":memory:")

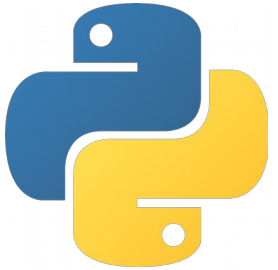
Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Hinweis
      - Alle großgeschriebenen Wörter sind Bestandteile der Sprache SQL. Allerdings unterscheidet SQL nicht zwischen Groß- und Kleinschreibung, weshalb wir auch alles hätten kleinschreiben können. Wegen der besseren Lesbarkeit werden wir SQL-Schlüsselwörter immer komplett groß- und von uns vergebene Namen durchgängig kleinschreiben

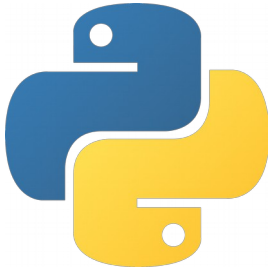
Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Für das Anlegen der Tabelle »Lager« sieht das SQL-Statement folgendermaßen aus

```
CREATE TABLE lager (
        fachnummer INTEGER,
        seriennummer INTEGER,
        komponente TEXT,
        lieferant TEXT,
        reserviert INTEGER
)
```

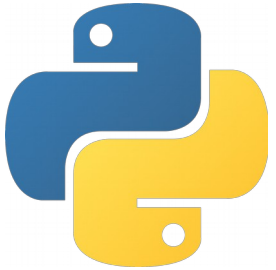
Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Folgende Tabelle zeigt das Umwandlungsschema von Python- zu SQLite-Datentypen

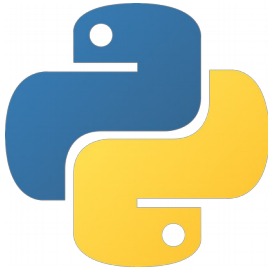
Python-Datentyp (Quelltyp)	SQLite-Datentyp (Zieltyp)
None	NULL
int	INTEGER
float	REAL
bytes (UTF8-Kodiert)	TEXT
str	TEXT
bytes	BLOB

Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Um mit der verbundenen Datenbank arbeiten zu können, werden sogenannte Cursor benötigt
    - Einen Cursor kann man sich ähnlich wie den blinkenden Strich in Textverarbeitungsprogrammen als aktuelle Bearbeitungsposition innerhalb der Datenbank vorstellen
    - Erst mit solchen Cursorn können wir Datensätze verändern oder abfragen, wobei es zu einer Datenbankverbindung beliebig viele Cursor geben kann

Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

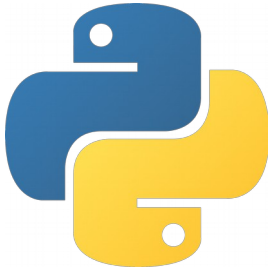
    - Ein neuer Cursor wird mithilfe der cursor-Methode des Connection-Objekts erzeugt

      ```
      cursor = connection.cursor()
      ```

    - Nun senden wir das SQL-Statement mithilfe der execute-Methode des Cursor-Objekts an die SQLite-Datenbank

      ```
      cursor.execute("""CREATE TABLE lager (
                        fachnummer INTEGER,
                        seriennummer INTEGER,
                        komponente TEXT,
                        lieferant TEXT,
                        reserviert INTEGER)""")
      ```

Python Datenspeicherung



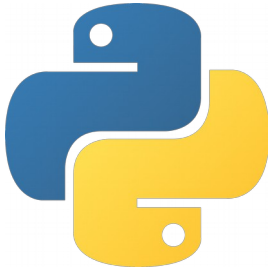
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Die Tabellen für die Lieferanten und Kunden erzeugen wir auf die gleiche Weise

```
cursor.execute("""CREATE TABLE lieferanten (
                        kurzname TEXT,
                        name TEXT,
                        telefonnummer TEXT)""")

cursor.execute("""CREATE TABLE kunden (
                        kundennummer INTEGER,
                        name TEXT,
                        anschrift TEXT)""")
```

Python Datenspeicherung



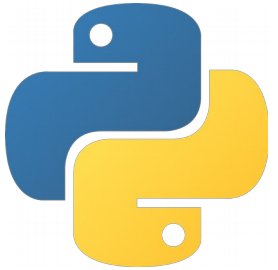
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Als Nächstes werden wir die noch leeren Tabellen mit unseren Beispieldaten füllen

    - Zum Einfügen neuer Datensätze in eine bestehende Tabelle dient das INSERT-Statement, das für den ersten Beispieldatensatz folgendermaßen aussieht

      ```
      INSERT INTO lager VALUES (
            1, 26071987, 'Grafikkarte Typ 1', 'FC', 0
      )
      ```

Python Datenspeicherung



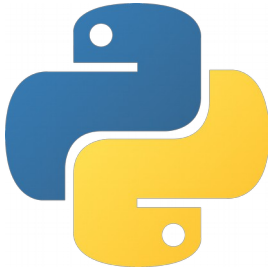
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Innerhalb der Klammern hinter VALUES stehen die Werte für jede einzelne Spalte in der gleichen Reihenfolge, wie auch die Spalten selbst definiert wurden

    - Wie bei allen anderen Datenbankabfragen auch können wir mit der execute-Methode unser Statement abschicken

      ```
      cursor.execute("""INSERT INTO lager VALUES (
                                  1, 26071987, 'Grafikkarte Typ 1', 'FC', 0)""")
      ```

Python Datenspeicherung



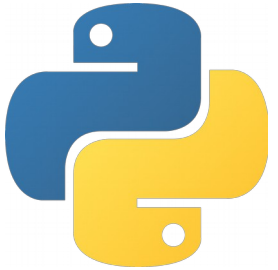
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Beim Einfügen von Datensätzen müssen Sie allerdings beachten, dass die neuen Daten nicht sofort nach Ausführen eines INSERT-Statements in die Datenbank daten geschrieben werden, sondern vorerst nur im Arbeitsspeicher liegen

    - Um sicherzugehen, dass die Daten wirklich auf der Festplatte landen und damit dauerhaft gespeichert sind, muss man die commit -Methode des Connection -Objekts aufrufen

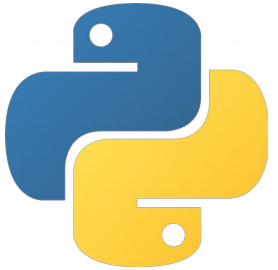
      connection.commit()

Python Datenspeicherung



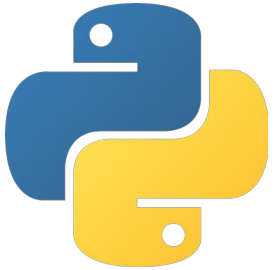
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Auf den ersten Blick scheint für solche Fälle die Formatierungsmethode format für Strings ein geeignetes Mittel zu sein
    - werte = (1, 26071987, "Grafikkarte Typ 1", "FC", 0)
    - "INSERT INTO lager VALUES ({0}, {1}, '{2}', '{3}', {4})".format(*werte)
    - Diese auf den ersten Blick elegante Methode entpuppt sich bei genauer Betrachtung aber als gefährliche Sicherheitslücke

Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Auf den ersten Blick scheint für solche Fälle die Formatierungsmethode format für Strings ein geeignetes Mittel zu sein
    - werte = (1, 26071987, "Grafikkarte Typ 1", "FC", 0)
    - "INSERT INTO lager VALUES ({0}, {1}, '{2}', '{3}', {4})".format(*werte)
    - Diese auf den ersten Blick elegante Methode entpuppt sich bei genauer Betrachtung aber als gefährliche Sicherheitslücke

Python Datenspeicherung



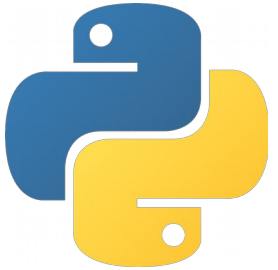
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Betrachten wir einmal die neue INSERT-Anweisung, die einen neuen Lieferanten in die Tabelle »Lieferanten« einfügen soll

      werte = ("DR", "Danger Electronics", "666'); Hier kann Schadcode stehen")

      "INSERT INTO lieferanten VALUES ('{0}', '{1}', '{2}')".format(*werte)

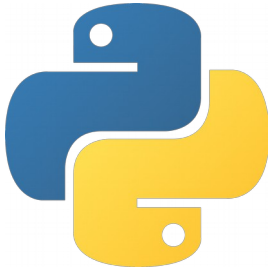
      'INSERT INTO lager VALUES ('DR', 'Danger Electronics', '666'); Hier kann Schadcode stehen')'

Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Der Wert für die Telefonnummer die den String '");" enthält, verunstaltet die SQL-Abfrage
    - Der Versuch, sie auszuführen, würde zu einem Fehler führen und damit unser Programm zum Absturz bringen
    - Durch den enthaltenen Text "Hier kann Schadcode stehen" soll angedeutet werden, dass es unter Umständen sogar möglich ist, eine Abfrage so zu manipulieren, dass wieder gültiger SQL-Code dabei herauskommt, wobei jedoch eine andere Operation als beabsichtigt (zum Beispiel das Auslesen von Benutzerdaten) ausgeführt wird

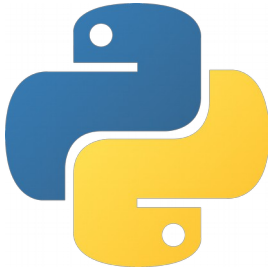
Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Hinweis
      - Verwenden Sie deshalb niemals die String-Formatierung zur Übergabe von Parametern in SQL-Abfragen!
    - Um sichere Parameterübergaben durchzuführen, schreibt man in den Query-String an die Stelle, an der der Parameter stehen soll, ein Fragezeichen und übergibt der execute-Methode ein Tupel mit den entsprechenden Werten als zweiten Parameter

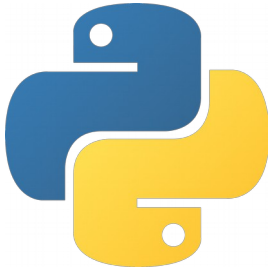
```
werte = ("DR", "Danger Electronics",
                                    "666'); Hier kann Schadcode stehen")
sql = "INSERT INTO lieferanten VALUES (?, ?, ?)"
cursor.execute(sql, werte)
```

Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - In diesem Fall kümmert sich SQLite darum, dass die übergebenen Werte korrekt umgewandelt werden und es zu keinen Sicherheitslücken durch böswillige Parameter kommen kann
    - Analog zur String-Formatierung gibt es auch hier die Möglichkeit, den übergebenen Parametern Namen zu geben und statt der tuple-Instanz mit einem Dictionary zu arbeiten
    - Dazu schreiben Sie im Query-String statt des Fragezeichens einen Doppelpunkt, gefolgt von dem symbolischen Namen des Parameters, und übergeben das passende Dictionary als zweiten Parameter an execute

Python Datenspeicherung



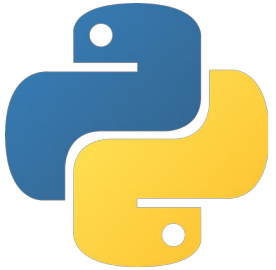
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

```
werte = {"kurz" : "DR", "name" : "Danger Electronics",
                "telefon" : "123456"}
sql = "INSERT INTO lieferanten VALUES (:kurz, :name,
:telefon)"
cursor.execute(sql, werte)
```

  - Mit diesem Wissen können wir unsere Tabellen elegant und sicher mit Daten füllen

```
for row in ((1, "2607871987", "Grafikkarte Typ 1", "FC", 0),
                        (2, "19870109", "Prozessor Typ 13", "LPE", 57),
                        (10, "06198823", "Netzteil Typ 3", "FC", 0),
                        (25, "11198703", "LED-Lüfter", "FC", 57),
                        (26, "19880105", "Festplatte 128 GB", "LPE", 12)):
    cursor.execute("INSERT INTO lager VALUES (?,?,?,?,?)", row)
```

Python Datenspeicherung

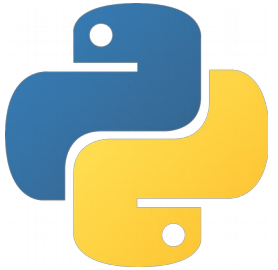


- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Strukturen wie die obige for-Schleife, die die gleiche Datenbankoperation sehr oft für jeweils andere Daten durchführen, kommen häufig vor und bieten großes Optimierungspotenzial

    - Aus diesem Grund haben cursor-Instanzen zusätzlich die Methode executemany, die als zweiten Parameter eine Sequenz oder ein anderes iterierbares Objekt erwartet, das die Daten für die einzelnen Operationen enthält

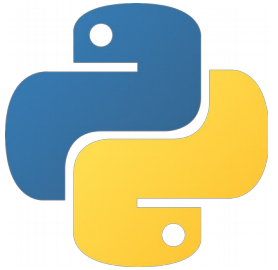
Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Wir nutzen executemany, um unsere Tabellen »Lieferanten« und »Kunden« mit Daten zu füllen

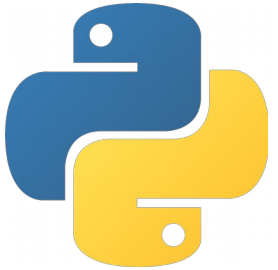
```
lieferanten = (("FC", "FiboComputing Inc.", "011235813"),
               ("LPE", "LettgenPetersErnesti", "026741337"),
               ("GC", "Golden Computers", "016180339"))
cursor.executemany("INSERT INTO lieferanten VALUES
                                                (?,?,?)",
                   lieferanten)
```

Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - kunden = ((12, "Heinz Elhurg", "Turnhallenstr. 1, 3763 Sporthausen"),
      (57, "Markus Altbert", "Kämperweg 24, 2463 Duisschloss"),
      (64, "Steve Apple", "Podmacstr 2, 7467 Iwarhausen"))
    - cursor.executemany("INSERT INTO kunden VALUES (?,?,?)", kunden)
    - Im nächsten Schritt wollen wir uns mit dem Abfragen von Daten beschäftigen

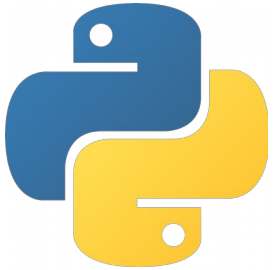
Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Abfrage
      SELECT lager.fachnummer, lager.komponente,
      lieferanten.name
      FROM lager, lieferanten
      WHERE lieferanten.telefonnummer='011235813' AND
      lager.lieferant=lieferanten.kurzname

Python Datenspeicherung

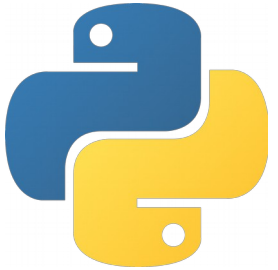


- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - SQLite

```
>>> sql = """
SELECT lager.fachnummer, lager.komponente,
                                                    lieferanten.name
FROM lager, lieferanten
WHERE lieferanten.telefonnummer='011235813' AND
                                        lager.lieferant=lieferanten.kurzname"""
>>> cursor.execute(sql)
>>> cursor.fetchall()
[(1, 'Grafikkarte Typ 1', 'FiboComputing Inc.'),
(10, 'Netzteil Typ 3', 'FiboComputing Inc.'),
(25, 'LED-Lüfter', 'FiboComputing Inc.')]
```

Python Datenspeicherung



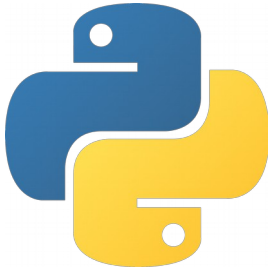
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Eine Abfrage mit cursor.fetchall liefert immer alle Ergebnisse der Abfrage auf einmal aus der Datenbank

    - Mit der Methode fetchone der cursor-Klasse fordern wir jeweils ein Ergebnis-Tupel (einen Datensatz) an

```
>>> cursor.execute("SELECT * FROM kunden")
>>> row = cursor.fetchone()
>>> while row:
        print(row)
        row = cursor.fetchone()
(12, 'Heinz Elhurg', 'Turnhallenstr. 1, 3763 Sporthausen')
(57, 'Markus Altbert', 'Kämperweg 24, 2463 Duisschloss')
(64, 'Steve Apple', 'Podmacstr 2, 7467 Iwarhausen')
```

Python Datenspeicherung



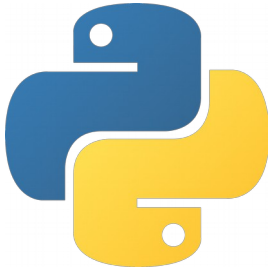
- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Das vorhergehende Beispiel ist aber etwas holprig, was den code angeht, daher auch selten verwendet

    - Der bessere Weg

```
>>> for row in cursor:
        print(row)
(12, 'Heinz Elhurg', 'Turnhallenstr. 1, 3763 Sporthausen')
(57, 'Markus Altbert', 'Kämperweg 24, 2463 Duisschloss')
(64, 'Steve Apple', 'Podmacstr 2, 7467 Iwarhausen')
```

Python Datenspeicherung

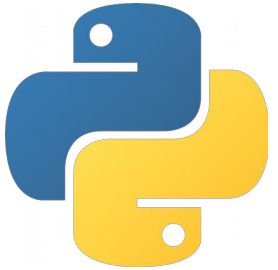


- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Die entsprechende Rückübersetzung von SQLite-Datentypen zu Python-Datentypen

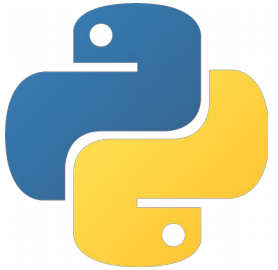
SQLite-Datentyp (Quelltyp)	Python-Datentyp (Zieltyp)
NULL	None
INTEGER	int
REAL	float
TEXT	str
BLOB	bytes

Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)
    - Das soll es als Einführung in den Umgang mit Datenbanken, mit Beispiel an SQLite in Python gewesen sein
    - Für weitere Ausführungen sollten Sie sich mit dem Thema Tiefergründig beschäftigen
    - Aufgabe
      Das hier als Grundaufbau dienende Beispiel in MySQL überführen, also statt SQLite → MySQL benutzen

Python Datenspeicherung



- Komprimierte Dateien lesen und schreiben-gzip
  - Datenbanken (Beispiel mit sqlite3 von Python mitgebracht)

    - Ansatz

```
>>> import mysql.connector
>>> con = mysql.connector.connect(user='<user>',
password='<passwd>', host='127.0.0.1', database='sakila')
>>> cursor = con.cursor()
>>> sql = '''SELECT * FROM actor WHERE
first_name='NICK';'''
>>> cursor.execute(sql)
>>> cursor.fetchall()
[(2, 'NICK', 'WAHLBERG', datetime.datetime(2006, 2, 15, 4,
34, 33)), (44, 'NICK', 'STALLONE', datetime.datetime(2006,
2, 15, 4, 34, 33)), (166, 'NICK', 'DEGENERES',
datetime.datetime(2006, 2, 15, 4, 34, 33))]
```