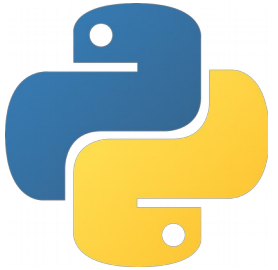
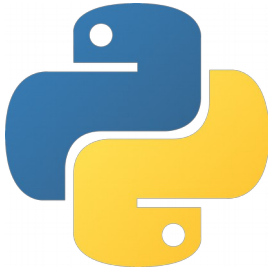
Python Datenspeicherung



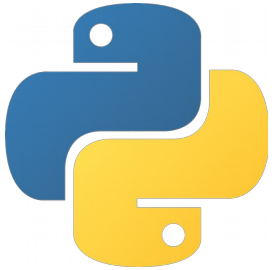
- Serialisierung von Instanzen – pickle
  - Das Modul pickle (dt. »pökeln«) bietet komfortable Funktionen für das Serialisieren von Objekten
  - Beim Serialisieren eines Objekts wird ein bytes-Objekt erzeugt, das alle Informationen des Objekts speichert, sodass es später wieder durch das sogenannte Deserialisieren rekonstruiert werden kann
  - Besonders für die dauerhafte Speicherung von Daten in Dateien ist pickle gut geeignet
  - Bei Klassen und Funktionen müssen Sie beachten, dass solche Objekte beim Serialisieren nur mit ihrem Klassennamen gespeichert werden

Python Datenspeicherung



- Serialisierung von Instanzen – pickle
  - Der Code einer Funktion oder die Definition der Klasse und ihre Attribute werden nicht gesichert
  - Wenn Sie also beispielsweise eine Instanz einer selbstdefinierten Klasse deserialisieren möchten, muss die Klasse in dem aktuellen Kontext genauso wie bei der Serialisierung definiert sein
  - Ist das nicht der Fall, wird ein UnpicklingError erzeugt
  - Das Modul pickle bietet seine Funktionalität über zwei Schnittstellen an
    - eine über die Funktionen dump und load
    - eine objektorientierte mit den Klassen Pickler und Unpickler

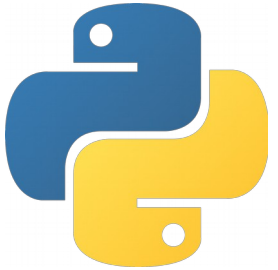
Python Datenspeicherung



- Serialisierung von Instanzen – pickle
  - Es gibt drei verschiedene Formate, in denen pickle seine Daten speichern kann
  - Jedes dieser Formate hat eine Zahl, um es zu identifizieren

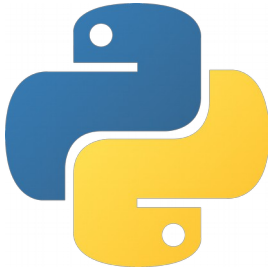
Nummer	Beschreibung
0	Der resultierende String besteht nur aus ASCII-Zeichen und kann deshalb auch von Menschen beispielsweise zu Debug-Zwecken gelesen werden. Das Protokoll 0 ist abwärtskompatibel mit älteren Python-Versionen.
1	Dieses Protokoll erzeugt einen Binärstring, der die Daten im Vergleich zur ASCII-Variante platzsparender speichert. Auch das Protokoll 1 ist abwärtskompatibel mit älteren Python-Versionen.
2	neues Binärformat, das besonders für Klasseninstanzen optimiert wurde Objekte, die mit dem Protokoll 2 serialisiert wurden, können nur von Python-Versionen ab 2.3 gelesen werden.
3	Ein neues Protokoll, das mit Python 3.0 eingeführt wurde und unter anderem auch den neuen bytes-Typ unterstützt. Die Daten können nicht mehr mit älteren Python-Versionen als 3.0 rekonstruiert werden. Trotzdem ist das Protokoll 3 das empfohlene und wird im pickle-Modul als Standard verwendet.

Python Datenspeicherung



- Serialisierung von Instanzen – pickle
  - Die Schnittstelle über dump und load
    - Anwendung
      import pickle
    - pickle.dump(obj, file[, protocol])
    - Diese Funktion schreibt die Serialisierung von obj in das Dateiobjekt file
    - Das übergebene Dateiobjekt muss dabei für den Schreibzugriff geöffnet worden sein
    - Mit dem Parameter protocol können Sie das Protokoll für die Speicherung übergeben

Python Datenspeicherung

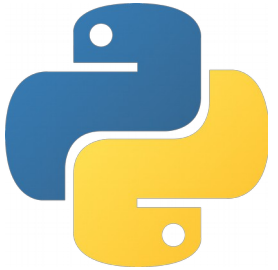


- Serialisierung von Instanzen – pickle
  - Die Schnittstelle über dump und load
    - Der Standardwert für protocol ist 3
    - Geben Sie ein Binärformat an, so muss das für file übergebene Dateiobjekt im binären Schreibmodus geöffnet worden sein

      ```
      >>> f = open("pickle-test.dat", "bw")
      >>> pickle.dump([1, 2, 3], f)
      ```

    - Für file können Sie neben echten Dateiobjekten jedes Objekt übergeben, das eine write-Methode mit einem String-Parameter implementiert, zum Beispiel StringIO-Instanzen

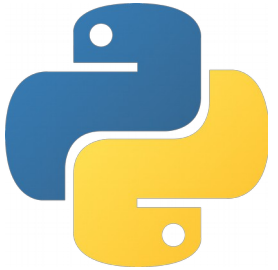
Python Datenspeicherung



- Serialisierung von Instanzen – pickle
  - Die Schnittstelle über dump und load
    - pickle.load(file)
    - Diese Funktion lädt, ausgehend von der aktuellen Leseposition des Dateiobjekts file, das nächste serialisierte Objekt
    - Dabei erkennt load selbstständig, in welchem Format die Daten gespeichert wurden

```
>>> f = open("pickle-test.dat", "rb")
>>> pickle.load(f)
[1, 2, 3]
```

Python Datenspeicherung



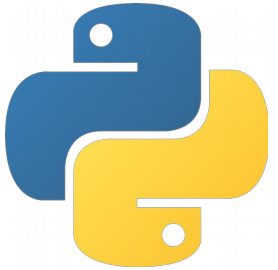
- Serialisierung von Instanzen – pickle
  - Die objektorientierte Schnittstelle Pickler und Unpickler

    - Gerade dann, wenn viele Objekte in dieselbe Datei serialisiert werden sollen, ist es lästig und schlecht für die Lesbarkeit, jedes Mal das Dateiobjekt und das zu verwendende Protokoll bei den Aufrufen von dump mit anzugeben

    - Neben den schon vorgestellten Modulfunktionen gibt es deshalb noch die beiden Klassen Pickler und Unpickler

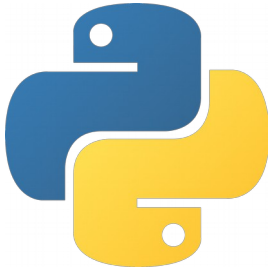
    - Pickler und Unpickler haben außerdem den Vorteil, dass Klassen von ihnen erben und so die Serialisierung anpassen können

Python Datenspeicherung



- Serialisierung von Instanzen – pickle
  - Die objektorientierte Schnittstelle Pickler und Unpickler
    - pickle.Pickler(file[, protocol])
    - Die Parameter file und protocol haben die gleiche Bedeutung wie bei der pickle.dump – Funktion
    - Das resultierende Pickler-Objekt hat eine Methode namens dump, die als Parameter ein Objekt erwartet, das serialisiert werden soll
    - Alle an die load-Methode gesendeten Objekte werden in das beim Erzeugen der Pickler-Instanz übergebene Dateiobjekt geschrieben

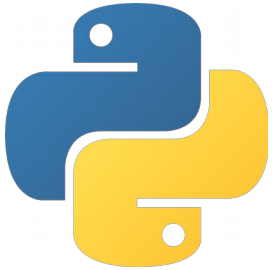
Python Datenspeicherung



- Serialisierung von Instanzen – pickle
  - Die objektorientierte Schnittstelle Pickler und Unpickler
    - pickle.Pickler(file[, protocol])

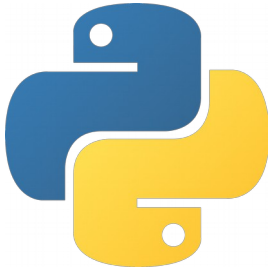
```
>>> p = pickle.Pickler(open("eine_datei.dat", "wb"), 2)
>>> p.dump({"vorname" : "Peter", "nachname" : "Kaiser"})
>>> p.dump([1, 2, 3, 4])
```

Python Datenspeicherung



- Serialisierung von Instanzen – pickle
  - Die objektorientierte Schnittstelle Pickler und Unpickler
    - pickle.Unpickler(file)
    - Das Gegenstück zu Pickler ist Unpickler, um aus der übergebenen Datei die ursprünglichen Daten wiederherzustellen
    - Unpickler-Instanzen besitzen eine parameterlose Methode namens load, die jeweils das nächste Objekt aus der Datei liest

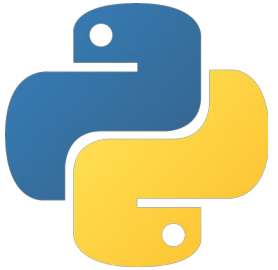
Python Datenspeicherung



- Serialisierung von Instanzen – pickle
  - Die objektorientierte Schnittstelle Pickler und Unpickler
    - pickle.Unpickler(file)

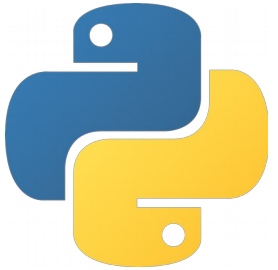
```
>>> u = pickle.Unpickler(open("eine_datei.dat", "rb"))
>>> u.load()
{'nachname': 'Kaiser', 'vorname': 'Peter'}
>>> u.load()
[1, 2, 3, 4]
```

Python Datenspeicherung



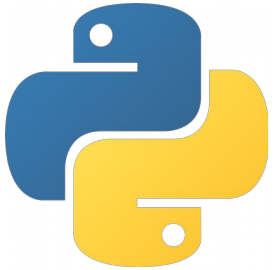
- Das Tabellenformat CSV – csv
  - Ein weit verbreitetes Import- und Exportformat für Datenbanken und Tabellenkalkulationen ist das CSV-Format (CSV steht für Comma Separated Values)
  - CSV-Dateien sind Textdateien, die zeilenweise Datensätze enthalten
  - Innerhalb der Datensätze sind die einzelnen Werte durch ein Trennzeichen wie beispielsweise das Komma voneinander getrennt, daher auch der Name
  - Die erste Zeile enthält die jeweiligen Spaltenköpfe, und alle folgenden Zeilen enthalten die eigentlichen Datensätze

Python Datenspeicherung



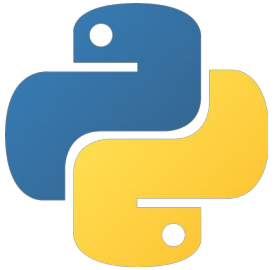
- Das Tabellenformat CSV – csv
  - Beispiel
    vorname,nachname,geburtsdatum,wohnort,haarfarbe
    Heinrich,Huhn,19.07.1980,Berlin,Braun
    Rudolf,Geier,19.09.1990,Dortmund,Braun
    Haken,Habicht,14.04.1959,Hamburg,Dunkelblond
    Edith,Falke,13.09.1987,Köln,Schwarz
    Rüdiger,Amsel,25.03.1988,München,Hellrot
  - Leider existiert kein Standard für CSV-Dateien, sodass sich beispielsweise das Trennzeichen von Programm zu Programm unterscheiden kann
  - Dieser Umstand erschwert es, CSV-Dateien von verschiedenen Quellen zu lesen, da immer auf das besondere Format der exportierenden Anwendung eingegangen werden muss

Python Datenspeicherung



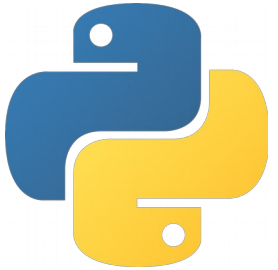
- Das Tabellenformat CSV – csv
  - Um trotzdem mit CSV-Dateien der verschiedensten Formate umgehen zu können, stellt Python das Modul csv zur Verfügung
  - Das csv-Modul implementiert reader - und writer-Klassen, die den Lese- bzw. Schreibzugriff auf CSV-Daten kapseln
  - Mithilfe sogenannter Dialekte kann dabei das Format der Datei angegeben werden
  - Standardmäßig gibt es vordefinierte Dialekte für die CSV-Dateien, die von Microsoft Excel generiert werden
  - Außerdem stellt das Modul eine Klasse namens Sniffer bereit, die den Dialekt einer Datei erraten kann

Python Datenspeicherung



- Das Tabellenformat CSV – csv
  - Verwendung
    import csv
  - Aufgabe
    Erstellen Sie sich eine ähnliche Datei, wie die hier angedeutete und benutzen Sie das Modul csv um die Datei aus zu lesen
    Erstellen Sie eine saubere Ausgabe der Daten

Python Datenspeicherung



- Temporäre Dateien – tempfile
  - Verwendung
    import tempfile
  - Wenn Ihre Programme umfangreiche Daten verarbeiten müssen, ist es oft nicht sinnvoll, alle Daten auf einmal im Arbeitsspeicher zu halten
  - Für diesen Zweck existieren temporäre Dateien, die es Ihnen erlauben, gerade nicht benötigte Daten vorübergehend auf die Festplatte auszulagern
  - Allerdings eigenen sich temporäre Dateien nicht dazu, Daten dauerhaft zu speichern
  - Sollte Sie dieses benötigen, dann schauen Sie sich das an