FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

DIE HOCHSCHULE, DIE MEHR KANN.

ILV Einführung in das Programmieren

Files

Mohamed Goha, BSc. | WS 2024/25

APPLIED LIFE SCIENCES | MASTERSTUDIENGANG BIOINFORMATIK

Files

> Reminder: Informationen werden in bits gespeichert

> Verschiedene Arten, Informationen zu codieren und decodieren (encode/decode): jpeg, mp3, pdf, …

> Bits können als Datei (file) auf einem Speichermedium gespeichert werden

File system und Dateiendung

- Das **file system** eines Betriebssystems (windows, unix, etc.) organisiert files
- Files haben einen Basisdateinamen (base filename)
- Um den Typ (Codierungs-/Decodierungsschema) unserem Betriebssystem anzuzeigen, können wir eine Dateiendung (file extension) verwenden.
  - (Typisches Format: ein_textfile.txt, ein_bild.jpg)

File system und Dateiendung

> Diese Dateiendung ist Teil des Dateinamens und dient nur als Hinweis (damit das Betriebssystem ein geeignetes Programm zum Öffnen der Datei wählt).

> Wir könnten z.B. ein_bild.jpg in ein_bild.txt umbenennen

Hierarchien

- Hierarchien sind betriebssystemabhängig:
  - In Unix wird das erste Verzeichnis als Root-Verzeichnis bezeichnet.
  - Unix verwendet das Zeichen “/” um Verzeichnisse zu trennen:
    - ***/ordner/unterordner/unterunterordner/einedatei.txt***
- In Windows ist das erste Verzeichnis die ID Ihres Laufwerks (oder der Partition), z.B. ***C:***

Hierarchien

> Windows verwendet das Zeichen “\”, um Verzeichnisse zu trennen:

C:\folder\subfolder\myfile.txt

Absolute und Relative Pfade

> Das Verzeichnis, in dem man sich zu einem Zeitpunkt befindet, wird **working directory** genannt.

> Der Speicherort einer Datei kann als **absoluter** oder **relativer** Pfad angegeben werden.

Absolute und Relative Pfade

- **Absolute Pfade**
  - enthalten das Root-Verzeichnis
  - sind unabhängig vom aktuellen Arbeitsverzeichnis
  - ***/home/user/ordner/meinedatei.txt***

Absolute und Relative Pfade

- **Relative Pfade**
  - Fangen bei einem gegebenen **working directory** an
  - Beispiel mit working directory ***"/home/user"***:
    - ***folder/myfile.txt***

Files in Python

- Files können mittels der built-in Funktion `open` geöffnet werden
- `filehandle = open(filename, mode)` filename = rel/abs. Pfad zur Datei
- filehandle: Objekt, das es uns ermöglicht, mit dem Dateiinhalt zu interagieren (es ist nicht der Dateiinhalt selbst!)
  - `filehandle.seek()`: Bewegt die Stream-Position (Startposition zum Lesen und Schreiben) an eine bestimmte Position in der Datei.
  - `filehandle.read()` : Inhalt der Datei lesen
  - `filehandle.write()` : Etwas in die Datei schreiben

Files in Python

> `filehandle = open(filename, mode)`

> mode:

  > “r”: “read only” - Aus Datei lesen, Error falls Datei nicht existiert

  > “w”: “write only” - schreibt in die Datei, erstellt eine neue Datei, wenn sie nicht existiert, oder löscht den ursprünglichen Dateiinhalt, wenn die Datei bereits existiert

Files in Python

> `filehandle = open(filename, mode)`

> mode:

  > "a": "append only" - schreibt in die Datei, erstellt eine neue Datei, wenn sie nicht existiert, behält aber die ursprünglichen Dateiinhalte bei und hängt neue Inhalte ans Ende der Datei an

  > "r+": "read and write" - liest aus der Datei und schreibt neue Inhalte an den Anfang der Datei, schlägt fehl, wenn die Datei nicht existiert

Files in Python

> `filehandle = open(filename, mode)`

> mode:

  > “a+”: “read and append” - liest aus der Datei und schreibt neue Inhalte ans Ende der Datei, erstellt eine neue Datei, wenn sie nicht existiert

> Default stream Positionen:

  > "r", "w", "r+": Anfang der Datei

  > "a", "a+": Ende der Datei

Files in Python

- `filehandle = open(filename, mode)`
- mode: spezifiziert auch, ob die Datei ein text file ist oder nicht
  - Text mode:
    - Inhalte werden als string Objekt(e) returned
    - String Objekte werden für das Schreiben in die Datei erwartet
    - mögliche modes: “r”, “w”, “a”, “r+”, ”a+” (die bisher angeführten modes)

Files in Python

- `filehandle = open(filename, mode)`
- mode: spezifiziert auch, ob die Datei ein text file ist oder nicht
  - Binary mode:
    - Inhalte werden als byte Objekt(e) returned
    - Byte Objekte werden für das Schreiben in die Datei erwartet
    - mögliche modes: “rb”, “wb”, “ab”, “rb+”, ”ab+”

Files öffnen und schließen

- Dateien die geöffnet werden, müssen wieder geschlossen werden
  - Die Inhalte die Sie in die Datei schreiben werden zuerst in einen **buffer** geschrieben und nicht unbedingt sofort in die Datei geschrieben, bis sie flushed und geschlossen wird.
  - Ihre Datei könnte beschädigt werden, wenn Sie das Programm beenden, bevor die Datei geschlossen wurde.

Files öffnen und schließen

- Wenn Ihr Programm abgebrochen wird (z.B. durch den Benutzer oder aufgrund von Exceptions), könnte die Datei nicht korrekt geschlossen werden.
- Wir könnten dieses Problem mit einem **try-finally**-Block lösen, in dem wir garantieren, dass die Datei ordnungsgemäß geschlossen wird.
- Oder wir verwenden die **with**-Anweisung, die die Datei automatisch schließt. Dies sollte, wenn möglich, verwendet werden. (kürzer und bequemer als **try-finally**).

Files öffnen und schließen: Beispiel

```
with open(filename, mode) as file:
    contents = file.read() # returns contents as a string
```

> Default mode: “r” (read)

> Beachten Sie, dass die **with**-Anweisung keine Exceptions behandelt, z.B. wenn die angegebene Datei fehlt.

> Sie stellt nur sicher, dass die Datei ordnungsgemäß geschlossen wird, falls sie zuvor geöffnet wurde.

Portabilität

- Das Nutzen von relativen Pfaden erhöht die Portabilität Ihres Codes:
  - Der Pfad ***/home/nutzer1/ordner/einedatei.txt*** könnte auf dem Rechner von Nutzer1 existieren, aber auf dem Rechner von Nutzer2 würde er nicht funktionieren.
  - Der Pfad ***ordner/einedatei.txt*** funktioniert sowohl auf dem Computer von Nutzer1 als auch von Nutzer2, solange sie vom richtigen Arbeitsverzeichnis aus starten (z.B. ***/home/nutzer1*** und ***/home/nutzer2/andererordner***).

Python Objekte in Dateien speichern

- Wir können nicht nur strings in Dateien speichern.
- **pickle - Modul:**
  - Ermöglicht es uns, viele Arten von Python-Objekten zu speichern und zu laden
  - Speichert Daten in Binärdateien und rekonstruiert sie beim erneuten Lesen
  - Kann viele verschiedene Python-Objekte verarbeiten

Python Objekte in Dateien speichern

- **dill - Modul**
  - Gleiche Schnittstelle wie pickle
  - Erweitert die Funktionalität von pickle
  - Kann mehr Arten von Python-Objekten speichern