Python für Linguisten

Dozentin: Wiebke Petersen & Co-Dozentin: Esther Seyffarth

Dateien schreiben und lesen
Tipps und Fragen zur 1. Hausaufgabe

print()

- Wir haben `print()` schon oft benutzt, um Text in der IDLE-Shell auszugeben.
- Dabei haben wir eine Menge optionaler Argumente der Funktion außer Acht gelassen:

```
>>> print(
        print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

  - `value`: Zeichenkette, die ausgegeben werden soll.
  - `...`: Ideen?
  - `sep`: String, der als Trennung zwischen einzelnen Zeichenketten gesetzt wird (standardmäßig ein Leerzeichen)
  - `end`: String, der nach der Ausgabe "implizit" eingefügt wird (standardmäßig ein Zeilenumbruch)
  - `file`: Ziel der Ausgabe (standardmäßig sys.stdout)
  - `flush`: Option zum Buffern bei großen Textmengen und Dateispeicherung (im Kurs nicht relevant)

Ausgabe in eine Datei

- Wir können nur dann das Argument `file` der `print`-Funktion verwenden, wenn wir vorher im Programm ein Datei-Objekt erstellen.

```
1 >>> myfile = open("myfile.txt")
2 >>> myfile
3 <_io.TextIOWrapper name='myfile.txt' mode='r' encoding='cp1252'>
```

- Die Dateiendung `.txt` sollten wir nicht vergessen, sie wird auch nicht automatisch ergänzt.
- Das Encoding cp1252 ist eine Windows-proprietäre Codierung und daher massiv problematisch!
- Der Modus `'r'` steht für "read", wir haben also noch keine Berechtigung, etwas in die Datei zu schreiben!

Öffnen oder Erstellen einer Datei

- Die wichtigsten Dateiöffnungsmodi für uns sind die folgenden:
  - `'r'` (read): Datei lesen.
  - `'w'` (write): Datei überschreiben. Dabei wird etwaiger Inhalt der Datei zuerst gelöscht!
  - `'a'` (append): Inhalt in Datei ergänzen, d.h. etwaiger Inhalt der Datei bleibt bestehen und unser Text wird ans Ende der Datei gesetzt.
- Das beste Encoding für stabile Programmierung ist UTF-8 (Unicode).

```
1 >>> myfile = open("myfile.txt", "w", encoding="utf8")
2 >>> myfile
3 <_io.TextIOWrapper name='myfile.txt' mode='w' encoding='utf8'>
```

Ausgabe in eine Datei

```
1 >>> myfile = open("myfile.txt", "w", encoding="utf8")
2 >>> print("In dieser Datei steht nun Text.", file=myfile)
3 >>> myfile.close()
```

- Wo liegt die Datei denn nun eigentlich...?
- Um sicherzugehen, können wir auch den absoluten Pfad zur Datei mit beim Dateinamen angeben. (Funktioniert nur, wenn der Pfad wohlgeformt ist und wir Schreibrechte dort haben.)
- Aufräumen ist wichtig! Wenn wir die Datei am Ende nicht schließen, ist nicht sichergestellt, dass unsere Ausgabe erfolgreich ist.

Übung zum Schreiben in Dateien

- Erstellen Sie ein Programm, das Kontaktdaten vom Nutzer abfragt und diese als Visitenkarte in einer Datei speichert.
- Der Name der Datei wird zusammengesetzt nach dem Muster `NachnameVorname.txt`.
- Alternativ können Sie dem Nutzer ermöglichen, den Namen für die Datei selbst einzugeben.
- Welche Benutzereingaben können zu Fehlern im Programm führen? Versuchen Sie, mindestens einen Error (im laufenden Programm) zu erzeugen!

Dateien lesen

- Den Dateiöffnungsmodus `'r'` verwenden wir, um eine existierende Datei zu öffnen und ihren Inhalt auszulesen.

```
1 >>> myfile = open("myfile.txt", "r", encoding="utf8")
2 >>> print(myfile)
3 <_io.TextIOWrapper name='myfile.txt' mode='r' encoding='utf8'>
```

- Um den Inhalt der Datei zu lesen, haben wir vier Möglichkeiten:
  - `myfile.read()`: Liest den gesamten Inhalt der Datei als einen einzigen String. Zeilenumbrüche sind im String als `\n` codiert.
  - `myfile.readline()`: Liest pro Aufruf jeweils die nächste Zeile der Datei als String. Der Zeilenumbruch wird am Ende der Zeile als `\n` im String repräsentiert. Falls keine Zeilen mehr übrig sind, ist die Rückgabe von `myfile.readline()` ein leerer String.
  - `myfile.readlines()`: Liest die gesamte Datei als eine Liste von Strings, wobei jede Zeile ein Element der Liste ist.
  - `for line in myfile: print(line)`

Verarbeitung von Text aus einer Datei

- Wenn wir Text aus einer Datei weiter verarbeiten wollen, müssen wir wissen, welches Format der Text in der Datei hat.
- Die Visitenkarten-Datei aus der letzten Aufgabe könnte etwa so aussehen:

```
Vorname:  Esther
Nachname:  Seyffarth
Mail:  esther.seyffarth@hhu.de
```

- Oder so:

```
Esther
Seyffarth
esther.seyffarth@hhu.de
```

- Oder so:

```
Vorname=Esther
Nachname=Seyffarth
Mail=esther.seyffarth@hhu.de
```

Verarbeitung von Text aus einer Datei

- Je nach Format des Textes in der Datei können wir mit der String-Operation `str.split(separator, count)` arbeiten.

```
1 >>> line_from_file = "Name = Esther"
2 >>> processed = line_from_file.split("=")
3 >>> print(processed)
4 ['Name ', ' Esther']
```

- Das Argument `count` ist optional. Wenn es angegeben wird, beschränkt es die Anzahl der Aufspaltungen des Strings. Lässt man es weg, wird der String so oft aufgeteilt wie möglich.

```
1 >>> some_string = "ein-Text-mit-vielen-Strichen"
2 >>> some_string.split("-", 2)
3 ['ein', 'Text', 'mit-vielen-Strichen']
```

- Tatsächlich ist auch das Argument `separator` optional. Lässt man es weg, wird an allen Whitespaces im String geteilt.
- Achtung: Die Teilstrings, die als Separator verwendet werden, gehen beim Aufspalten des Strings verloren.

Verarbeitung von Text aus einer Datei

```
1 >>> print(processed)
2 ['Name ', ' Esther']
```

- Die Teilstrings enthalten im Beispiel noch Whitespaces, die wir für die weitere Verarbeitung nicht brauchen.
- Wir können sie mit dem Befehl `token = processed[0].strip()` entfernen.
- `str.strip()` entfernt alle führenden und abschließenden Whitespaces. Wir können es auch mit einem String-Argument aufrufen:

```
1 >>> sweary_string = "&#$!text%+*"
2 >>> sweary_string.strip("#+*$&!%")
3 'text'
```

- Was fällt Ihnen auf?

Weitere nützliche Stringmanipulationen

- Wir kennen bereits:
  - `string[2]`
  - `if substring in string: ...`
  - `string.split(separator)`
  - `string.strip(padding)`
- Weitere nützliche Operationen für Strings:
  - `string.startswith(prefix)`, `string.endswith(suffix)`
  - `string.count(substring)`
  - `string.lower()`, `string.upper()`
  - `string.capitalize()`, `string.title()`
  - `string.isupper()`, `string.islower()`, `string.isalnum()`
  - `string.join(stringliste)`
  - `string.replace(old, new)`
- Erklärungen und Verwendungsbeispiele für alle diese Funktionen finden Sie unter:
  `https://docs.python.org/3/library/stdtypes.html#str`
  (Abschnitt "String Methods")

Übungsaufgaben zu Dateiverarbeitung

- Schreiben Sie ein Programm, das die Wörter (Tokens) in einer Datei zählt.
- Schreiben Sie ein Programm, das die Anzahl der **verschiedenen** Wörter (Types) in einer Datei ermittelt. (Welche Datenstruktur eignet sich hierfür?)
- Schreiben Sie ein Programm, das eine Datei zensiert: Alle Zahlen in der Datei werden durch `x` ersetzt. Achtung: Erstellen Sie für die Ausgabe eine neue Datei, der man ansieht, dass sie die zensierte Version des Textes enthält.

Erweiterung der Hausaufgabe

- Wenn Sie Teil 3 der Aufgabe geschafft haben, versuchen Sie, die Ausgabe in eine Datei umzuleiten!

Tipps zur Hausaufgabe

- Das Schwierigste am Lösen von komplexeren Programmieraufgaben ist die Planung.
- Schreiben Sie nicht drauflos! Überlegen Sie sich genau:
  - In welche Sinnabschnitte lässt das Programm sich aufteilen?
  - Welche Vorgänge müssen in welcher Reihenfolge erfolgen?
- Oft hilft es auch, das Grundgerüst des Programms ohne die feineren Details zuerst zu schreiben - sobald Ihr Programm ausgeführt werden kann, können Sie die komplexeren Komponenten entwickeln.
- Mögliche Startpunkte für Hausaufgabe 1:
  - Zunächst nur die übergebenen Parameter der Funktion `make_wordforms()` ausgeben.
  - Zunächst nur die Namen der 5 Fälle ausgeben, unabhängig von den Parametern.
  - ...