Skriptsprachen: Python
Daten und Dateien

Jan Krüger, Alexander Sczyrba

Technische Fakultät
Universität Bielefeld

12. Februar 2018

Python

–

Nutzereingaben, Nutzerausgaben

Kommandozeilenparameter, Ein-/Ausgabe von Nutzerdaten

Kommandozeilenparameter – `argparse`-Modul

Bisher könnt ihr Kommandozeilenparameter lediglich direkt über `sys.argv` verarbeiten. `argparse` bietet einen sehr eleganten objektorientierten Ansatz zum Verarbeiten dieser mit

- autegenerierter Hilfe für den Nutzer
- positionalen und optionalen Parametern
- Parametergruppen
- Typcheck
- objektorientem Zugriff

Parser erstellen $\rightarrow$ Argumente und Gruppen hinzufügen $\rightarrow$ Parsen $\rightarrow$ ggf. Fehlerausgabe nach Check $\rightarrow$ Argumente verarbeiten

Konstruktor `prsr = argparse.ArgumentParser()`

Argument Wenn der Name des Arguments mit einem oder mehreren „-“ beginnt, so ist er optional, ansonsten positional und somit Pflicht.

```
prsr.add_argument(ANAME[,
 LONGANAME, help="...", type=T,
  action="..."])
```

Ausschließende Gruppe Sich ausschließende Parameter werden gruppiert.

```
grp = prsr.add_mutually_exclusive_group()
grp.add_argument(...)
```

Parsen `args = prsr.parse_args()`

argparse-Modul – Beispiel

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("square",
  help="display␣a␣square␣of␣a␣given␣number",
  type=int)
args = parser.parse_args()
print args.square**2
```

```
$ python prog.py 4
16
$ python prog.py four
usage: prog.py [-h] square
prog.py: error: argument square: invalid int value: 'four'
```

Das `args`-Obejkt

Der Aufruf der `parse_args()`-Methode bewirkt:

- Typ-Check aller Parameter
- Check auf Vorhandensein aller positionaler Parameter
- Erfüllllbarkeit der gruppierten Parameter
- Anlegen des Rückgabeobjekts, wobei jedem Argument ein Feld zugeordnet ist, auf das im weiteren Verlauf per `args.NAME` zugegriffen werden kann

argparse-Modul – Beispiel

```
import argparse

parser = argparse.ArgumentParser(description="calculate␣X^Y")
parser.add_argument("x", type=int, help="the␣base")
parser.add_argument("y", type=int, help="the␣exponent")

group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true")
group.add_argument("-q", "--quiet", action="store_true")

args = parser.parse_args()
answer = args.x**args.y

if args.quiet:
    print answer
elif args.verbose:
    print "{}␣to␣the␣power␣{}␣equals␣{}".format(args.x,
      args.y, answer)
else:
    print "{}^{}␣==␣{}".format(args.x, args.y, answer)
```

Übung – `argparse` + `Fasta`

Erweitert euer Fasta-Generierungsskript um die Möglichkeit, mit Kommandozeilenparametern umgehen zu können.

Diese Schalter/Argumente sollen implementiert werden:

- LENGTH: Länge der Fasta-Sequenz
- NAME: Name des Datensatzes (nach > kommend)
- -d, -r, -aa: Ausschließende Gruppe mit:
  - -d: DNA-Alphabet (ATCG)
  - -r: RNA-Alphabet (AUCG)
  - -aa: Aminosäurealphabet (ACDEFGHIKLMNPQRSTVWY)

Interaktive Eingabe – `rawinput()`

Um Eingaben während der Abarbeitung eines Pytohn-Skripts einlesen zu können, kann die `raw_input([prompt])`-Methode verwendet werden.

```
> s = raw_input("Filmname: ")
Filmname: Snakes on a Plane
> s
"Snakes on a Plane"
```

Die Methode gibt `prompt` in STDOUT aus und warten auf Eingabe des Benutzers. Die Eingabe wird durch Zeilenende beendet und als `string` zurückgeliefert (gestrippt).

Ausgabe von Variablen

Bisher vernachlässigt: Ausgabe von mehreren Variablen als String bzw. die Konkatenation dieser.

```
> z = 123
> print z+"ist␣durch␣"+3+"teilbar."
TypeError: cannot concatenate 'str' and
  'int' objects
```

Abhilfe: `str()` und `repr()`

```
> print str(z)+"␣ist␣durch␣"+str(3)+"␣teilbar."
123 ist durch 3 teilbar.
```

Ausgabe von Variablen

str(<V>) Rückgabewert: menschen-lesbare Stringrepräsentation

repr(<V>) Rückgabewert: maschinen-lesbarer String (Interpreter-verarbeitbar)

```
> str(s)
'Hello, world.'
> repr(s)
"'Hello, world.'"
> str(1.0/7.0)+" "+repr(1.0/7.0)
'0.142857142857 0.14285714285714285'
```

Ausgabe von Variablen – String-Formatierung

Mittels `str.format()` kann die Anzeige von Strings bzw. die Formatierung entsprechend verändert werden. `{}` sind Platzhalter.

Automatisch

```
> print '{} and {}'.format('spam', 'eggs', 42)
spam and eggs
```

Positional

```
> print '{0} and {1}'.format('spam', 'eggs', 42)
spam and eggs
> print '{1} and {2}'.format('spam', 'eggs', 42)
eggs and 42
```

Kein Mix aus beiden Notationen erlaubt!

Ausgabe von Variablen – String-Formatierung

Eine dritte Möglichkeit besteht in der Verwendung von *keywords*, die beliebig mit einem der anderen Verfahren genutzt werden kann.

```
> print 'This␣{food}␣is␣{adjective}.'.format(
  food='spam', adjective='absolutely␣horrible')
This spam is absolutely horrible.

> print 'The␣story␣of␣{0},␣{other}␣and␣{1}.'.
  format('Bill', 'Manfred', other='Georg')
The story of Bill, Georg, and Manfred.
```

Wichtig hierbei: Die *keywords* müssen nach den „normalen“ Variablen stehen!

Ausgabe von Variablen – String-Formatierung

Die Genauigkeit von Zahlen kann in folgender Art formatiert werden: `{P:C[V.N|V]F}`:

- `P` Position
- `C` Flags für Vorzeichen, Auffüllmethode
- `V` Aufzufüllende Stellen nach links
- `:` Einleitendes Formatierungszeichen
- `.N` Nachkommagenauigkeit bei Fließkommazahlen
- `F` Variablentyp

```
> print '{:+08d} PI is ~{:7.3f}.'.
    format(1, math.pi)
+0000001 PI is ~  3.142.
|       |         |     |
01234567          1234567
```

Details: String Formatting-Doku

Python

–

Daten lesen und schreiben

Dateien, Streams

Dateizugriff

Die `open(<FILENAME>[, <MODE>])`-Methode liefert als Rückgabewert ein `file`-Objekt, auf dem Standardmethoden zum Lesen und Schreiben definiert sind.

`FILENAME` gibt den Namen der zu öffnenden Datei an (Default: relative Pfadangebe). Der Standardmodus ist lesend, alternativ kann ein `MODE` (`string`) angegeben werden:

r	nur lesend – default	w	nur schreibend
r+	lesend und schreibend	a	anhängend schreibend

Dateizugriff – lesen

Drei Standardmethoden zum Einlesen einer Datei:

read([size])	Vollständig einlesen, sonst `size` Bytes. Rückgabetyp: `string`
readlines()	Vollständig einlesen. Rückgabetyp: `list` von `strings`
readline()	Einzelne Zeile einlesen. Rückgabetyp: `string`

Zu beachten ist, dass

- Zeilenenden nicht entfernt werden.
- sofern keine Zeilen mehr eingelesen werden können, der Rückgabewert stets ein leerer String ist.
- abschließend *immer* `close()` aufgerufen werden sollte.

Dateizugriff – lesen

Hier ein Codebeispiel mit Loops und Fehlerbehandlung

```
> for line in f:
>    print line,
This is the first line of the file.
Second line of the file

> with open('workfile', 'r') as f:
>    read_data = f.read()
> f.closed
True
```

Dateizugriff – schreiben

Zwei Standardmethoden zum Schreiben in eine Datei:

write(<STRING>) `STRING` wird in die Datei geschrieben

writelines(<SEQ<STRING») Alle Strings, die in der übergebenen Sequenz stehen, werden in die Datei geschrieben.

Zu beachten ist, dass

- Zeilenenden nicht automatisch hinzugefügt werden.
- `writelines()` als Parameter eine Sequenz von Strings erwartet.
- `write()`/`writelines()` lediglich Strings verarbeiten können!

Dateizugriff – schreiben

```
> f = open('the.question', 'w')
> f.readline()
''

> value = ('the␣answer', 42)
> s = str(value)
> f.write(s)

> f.seek(0)
> f.readline()
"('the␣answer',␣42)"
> f.readline()
''
> f.close()
```

Dateizugriff – STDIN

Der Zugriff auf den Inhalt von STDIN erfolgt auf dieselbe Art und Weise, wie bei einem standard file-Objekt.

Hierfür muss das Modul sys importiert werden, worauf das Objekt sys.stdin zur Verfügung steht, dass alle lesenden Methoden von file besitzt.

```
> # count.py
> import sys
> data = sys.stdin.readlines()
> print "Counted", len(data), "lines."

$ cat count.py | ./count.py
Counted 3 lines.
```

Übung – Fasta Im-/Export

Erweitert euer parametrisiertes Fasta-Programm um folgende Funktionalität:

1. Dateiexport, wenn ein Dateiname als zusätzlicher Parameter angegeben wurde
2. Einlesen einer Datei über STDIN, in der ein neues Alphabet festgelegt ist. Hierbei soll in jeder Zeile ein neues Zeichen stehen. Das Alphabet `{A, U, 1, .}` wird bspw. durch folgende Datei definiert:

```
1| A
2| U
3| 1
4| .
```

3. Verwendet `sys.stdin.isatty()` zum STDIN-Check.