Vorlesung 10

3. Objekt-orientierte Programmierung (erste Schritte)

3.1 Benutzung von Daten-Typen

- Datentyp für Farben: `Color`
- Wetterdaten als buntes Streifendiagramm darstellen
- Datentyp für Bilder: `Picture`
- Datentyp für Assoziative Arrays: `dict`
- Morsecode übersetzen
- Mehr über den Datentyp `str` und ein Exkurs zu `pydoc`
- Schreiben wie Goethe
- Docstrings für `pydoc`

4 Dictionaries – der Datentyp `dict`

Assoziative Arrays

Bei Arrays hat man eine Zuordnung von Indizes zu Werten.

Bei Dictionaries hat man eine Zuordnung von *Schlüsseln* zu Werten.

Die Werte können über die Schlüssel „angesprochen“ werden.

Die Schlüssel können von einem grundlegenden Datentyp oder Tupel davon etc. sein („hashable“).

Wörterbuch:

Adler	eagle
Buch	book
Dienst	service
Efeu	ivy
funkeln	to sparkle
⋮	⋮

Python:

```
d = dict()          # d ist ein leeres Dictionary.
d['Adler'] = 'eagle'
d['Buch'] = 'book'
d['Dienst'] = 'service'
d['Efeu'] = 'ivy'
d['funkeln'] = 'to sparkle'
...
```

Benutzung des Datentyps `dict` – Teil des API

Erzeugen eines Dictionary:

$\{s_1 : w_1, s_2 : w_2, ..., s_n : w_n\}$	ein Dictionary der Länge n mit den Zuordnungen Schlüssel s_1 zu Wert w_1, ..., Schlüssel s_n zu Wert w_n
`dict()` oder $\{\}$	ein leeres Dictionary (ohne Zuordnungen)

Zugriff auf ein Dictionary `d`:

`d[s]`	der Wert, der Schlüssel `s` in `d` zugeordnet ist	(indizierter Zugriff)
`d[s] = x`	dem Schlüssel `s` wird Wert `x` zugeordnet	(indizierte Zuweisung)
`del(d[s])`	lösche Schlüssel `s` aus `d`	
`s in d`	ergibt `True`, falls Schlüssel `s` im Dictionary `d` vorkommt, sonst `False`	
`for s in d:`	iteriere über alle Schlüssel `s` von `d`	(Durchlaufen)
`len(d)`	die Anzahl der Schlüssel von `d`	
`d.keys()`	ein Array aus allen Schlüsseln von `d`	
`d.values()`	ein Array aus allen Werten von `d`	
`d.items()`	ein Array aus Tupeln, die jeweils aus einem Schlüssel und dem zugehörigen Wert von `d` bestehen	

5 Programmierauftrag: übersetze in und aus dem Morsecode

Wir wollen ein Programm schreiben,

- das ein Eingabewort in das Morsealphabet kodiert
- oder aus dem Morsealphabet dekodiert.

Das Morsealphabet ordnet jedem Großbuchstaben eine Zeichenfolge aus Punkten und Strichen zu, z.B.

A .-	D -..	G --.	J .---
B -...	E .	H	K -.-
C -.-.	F ..-.	I ..	L .-..

Das Wort KABEL wird zu -.- .- -... . .-.. kodiert

und das Wort -... . .. ..-. .- .-.. .-.. wird zu dekodiert.

5 Programmierauftrag: übersetze in und aus dem Morsecode

Wir wollen ein Programm schreiben,

- das ein Eingabewort in das Morsealphabet kodiert
- oder aus dem Morsealphabet dekodiert.

Das Morsealphabet ordnet jedem Großbuchstaben eine Zeichenfolge aus Punkten und Strichen zu, z.B.

A .-	D -..	G --.	J .---
B -...	E .	H	K -.-
C -.-.	F ..-.	I ..	L .-..

Das Wort `KABEL` wird zu `-.- .- -... . .-..` kodiert
und das Wort `-... . .. ..-. .- .-.. .-..` wird zu `BEIFALL` dekodiert.

Teilaufgaben und Daten

Programm:

- Übersetzung eines Wortes in den Morsecode
- Übersetzung eines Wortes aus dem Morsecode

- Der Morsecode muss in das Programm kommen.

Daten:

- ein Dictionary mit Buchstaben als Schlüsseln und Morsezeichen als Werten (Übersetzung in den Morsecode, kodieren)
- ein Dictionary mit Morsezeichen als Schlüsseln und Buchstaben als Werten (Übersetzung aus dem Morsecode, dekodieren)

Die grobe Struktur des Programmes und der Daten

Datei `morsealphabet.txt`:

```
A  .-
B  -...
C  -.-.
...
```

1. Lies den Morsecode aus einer Datei ein und erzeuge
ein Dictionary zum Kodieren und
ein Dictionary zum Dekodieren
daraus.

Kodierungs-Dictionary:

```
{ 'A': '.-', 'B': '-...', 'C': '-.-.', ... }
```

Dekodierungs-Dictionary:

```
{ '.-': 'A', '-...': 'B', '-.-.': 'C', ... }
```

2. Lies die Eingabe.

2.1 Falls sie aus Buchstaben besteht: kodiere sie mit Hilfe des Kodierungs-Dictionaries.

2.2 Falls sie aus Morsezeichen besteht: dekodiere sie mit Hilfe des Dekodierungs-Dictionaries.

Teil 1: Morsealphabet einlesen und in Dictionaries eintragen

Datei	`zeile in eingabedatei`	`z = str.split(zeile)`	Eintrag in `codeDict` und `decodeDict`
`A`	`.-`		
`B`	`-...`		
	⋮		

`codeDict` ist ein Dictionary für
die Zuordnung von Buchstaben zu Morsezeichen.

`decodeDict` ist ein Dictionary für
die Zuordnung von Morsezeichen zu Buchstaben.

Dictionary `codeDict`	
Schlüssel	Wert

Dictionary `decodeDict`	
Schlüssel	Wert

Teil 1: Morsealphabet einlesen und in Dictionaries eintragen

Datei	`zeile in eingabedatei`	`z = str.split(zeile)`	Eintrag in `codeDict` und `decodeDict`
`A .-`	`'A .-'`		
`B -...`			
⋮			

`codeDict` ist ein Dictionary für
die Zuordnung von Buchstaben zu Morsezeichen.

`decodeDict` ist ein Dictionary für
die Zuordnung von Morsezeichen zu Buchstaben.

Dictionary `codeDict`	
Schlüssel	Wert

Dictionary `decodeDict`	
Schlüssel	Wert

Teil 1: Morsealphabet einlesen und in Dictionaries eintragen

Datei	`zeile in eingabedatei`	`z = str.split(zeile)`	Eintrag in `codeDict` und `decodeDict`
`A .-`	`'A .-'`	`['A', '.-']`	
`B -...`			
⋮			

`codeDict` ist ein Dictionary für
die Zuordnung von Buchstaben zu Morsezeichen.

`decodeDict` ist ein Dictionary für
die Zuordnung von Morsezeichen zu Buchstaben.

Dictionary `codeDict`	
Schlüssel	Wert

Dictionary `decodeDict`	
Schlüssel	Wert

Teil 1: Morsealphabet einlesen und in Dictionaries eintragen

Datei	`zeile in eingabedatei`	`z = str.split(zeile)`	Eintrag in `codeDict` und `decodeDict`
`A .-`	`'A .-'`	`['A', '.-']`	`codeDict['A'] = '.-'` `decodeDict['.-'] = 'A'`
`B -...`			
⋮			

`codeDict` ist ein Dictionary für
die Zuordnung von Buchstaben zu Morsezeichen.

`decodeDict` ist ein Dictionary für
die Zuordnung von Morsezeichen zu Buchstaben.

Dictionary `codeDict`	
Schlüssel	Wert
`'A'`	`'.-'`

Dictionary `decodeDict`	
Schlüssel	Wert
`'.-'`	`'A'`

Teil 1: Morsealphabet einlesen und in Dictionaries eintragen

Datei	`zeile in eingabedatei`	`z = str.split(zeile)`	Eintrag in `codeDict` und `decodeDict`
`A .-`	`'A .-'`	`['A', '.-']`	`codeDict[z[0]] = z[1]` `decodeDict[z[1]] = z[0]`
`B -...`			
⋮			

`codeDict` ist ein Dictionary für
die Zuordnung von Buchstaben zu Morsezeichen.

`decodeDict` ist ein Dictionary für
die Zuordnung von Morsezeichen zu Buchstaben.

Dictionary `codeDict`	
Schlüssel	Wert
`'A'`	`'.-'`

Dictionary `decodeDict`	
Schlüssel	Wert
`'.-'`	`'A'`

Teil 1: Morsealphabet einlesen und in Dictionaries eintragen

Datei	`zeile in eingabedatei`	`z = str.split(zeile)`	Eintrag in `codeDict` und `decodeDict`
`A .-`	`'A .-'`	`['A', '.-']`	`codeDict[z[0]] = z[1]` `decodeDict[z[1]] = z[0]`
`B -...`	`'B -...'`		
⋮			

`codeDict` ist ein Dictionary für
die Zuordnung von Buchstaben zu Morsezeichen.

`decodeDict` ist ein Dictionary für
die Zuordnung von Morsezeichen zu Buchstaben.

Dictionary `codeDict`	
Schlüssel	Wert
`'A'`	`'.-'`

Dictionary `decodeDict`	
Schlüssel	Wert
`'.-'`	`'A'`

Teil 1: Morsealphabet einlesen und in Dictionaries eintragen

Datei	`zeile in eingabedatei`	`z = str.split(zeile)`	Eintrag in `codeDict` und `decodeDict`
`A .-`	`'A .-'`	`['A', '.-']`	`codeDict[z[0]] = z[1]` `decodeDict[z[1]] = z[0]`
`B -...`	`'B -...'`	`['B', '-...']`	
⋮			

`codeDict` ist ein Dictionary für
die Zuordnung von Buchstaben zu Morsezeichen.

`decodeDict` ist ein Dictionary für
die Zuordnung von Morsezeichen zu Buchstaben.

Dictionary `codeDict`	
Schlüssel	Wert
`'A'`	`'.-'`

Dictionary `decodeDict`	
Schlüssel	Wert
`'.-'`	`'A'`

Teil 1: Morsealphabet einlesen und in Dictionaries eintragen

Datei	`zeile in eingabedatei`	`z = str.split(zeile)`	Eintrag in `codeDict` und `decodeDict`
`A .-`	`'A .-'`	`['A', '.-']`	`codeDict[z[0]] = z[1]`
			`decodeDict[z[1]] = z[0]`
`B -...`	`'B -...'`	`['B', '-...']`	`codeDict[z[0]] = z[1]`
⋮			`decodeDict[z[1]] = z[0]`

`codeDict` ist ein Dictionary für
die Zuordnung von Buchstaben zu Morsezeichen.

`decodeDict` ist ein Dictionary für
die Zuordnung von Morsezeichen zu Buchstaben.

Dictionary `codeDict`	
Schlüssel	Wert
`'A'`	`'.-'`
`'B'`	`'-...'`

Dictionary `decodeDict`	
Schlüssel	Wert
`'.-'`	`'A'`
`'-...'`	`'B'`

Teil 1: Morsealphabet einlesen und in Dictionaries eintragen

Datei	`zeile in eingabedatei`	`z = str.split(zeile)`	Eintrag in `codeDict` und `decodeDict`
`A .-`	`'A .-'`	`['A', '.-']`	`codeDict[z[0]] = z[1]`
			`decodeDict[z[1]] = z[0]`
`B -...`	`'B -...'`	`['B', '-...']`	`codeDict[z[0]] = z[1]`
⋮			`decodeDict[z[1]] = z[0]`

`codeDict` ist ein Dictionary für die Zuordnung von Buchstaben zu Morsezeichen.

`decodeDict` ist ein Dictionary für die Zuordnung von Morsezeichen zu Buchstaben.

Dictionary `codeDict`

Schlüssel	Wert
`'A'`	`'.-'`
`'B'`	`'-...'`

Dictionary `decodeDict`

Schlüssel	Wert
`'.-'`	`'A'`
`'-...'`	`'B'`

API:

Funktion	Bedeutung
`uebersetzerErzeugen()`	liest das Morsealphabet aus der Datei `morsealphabet.txt`. Sie gibt ein Dictionary mit der Zuordnung Buchstaben→Morsezeichen und ein Dictionary mit der Zuordung Morsezeichen→Buchstaben zurück.

Teil 2.1: Wort aus Großbuchstaben in den Morsecode kodieren

Grundidee beim Kodieren eines Buchstabens:

der Buchstabe x hat Kodierung `codeDict[`x`]`

Grundidee beim Kodieren eines Wortes:

schreibe die Kodierung jedes Buchstabens des Wortes hintereinander in einen String

Teil 2.1: Wort aus Großbuchstaben in den Morsecode kodieren

Grundidee beim Kodieren eines Buchstabens:

der Buchstabe x hat Kodierung `codeDict[`x`]`

Grundidee beim Kodieren eines Wortes:

schreibe die Kodierung jedes Buchstabens des Wortes hintereinander in einen String

API:

Funktion	Bedeutung
`kodiere(s, c)`	`s` ist ein String aus Großbuchstaben, `c` ist ein Dictionary mit der Zuordnung Buchstaben→Morsezeichen. Rückgabewert ist die Kodierung von `s` im Morsecode (`str`).

Teil 2.2: Wort aus Morsezeichen in Buchstaben dekodieren

Grundidee zum Dekodieren eines Morsezeichens:

das Morsezeichen m hat Kodierung `decodeDict[`m`]`

Grundidee zum Dekodieren einer Folge von Morsezeichen:

schreibe die Dekodierung jedes Morsezeichens hintereinander in einen String

Teil 2.2: Wort aus Morsezeichen in Buchstaben dekodieren

Grundidee zum Dekodieren eines Morsezeichens:

das Morsezeichen m hat Kodierung `decodeDict[`m`]`

Grundidee zum Dekodieren einer Folge von Morsezeichen:

schreibe die Dekodierung jedes Morsezeichens hintereinander in einen String

API:

Funktion	Bedeutung
`dekodiere(m, c)`	`m` ist ein String aus Morsezeichen und Leerzeichen, `c` ist ein Dictionary mit der Zuordnung Morsezeichen→Buchstaben. Rückgabewert ist die Dekodierung von `m` aus dem Morsecode (`str`).

Teil 2: Eingabe kodieren bzw. dekodieren

Ob man kodieren oder dekodieren soll,
erkennt man am ersten Zeichen der Eingabe:

Wenn es . oder – ist, dann muss die Eingabe dekodiert werden,
und sonst muss sie kodiert werden.

Teil 2: Eingabe kodieren bzw. dekodieren

Ob man kodieren oder dekodieren soll,
erkennt man am ersten Zeichen der Eingabe:

Wenn es `.` oder `-` ist, dann muss die Eingabe dekodiert werden,
und sonst muss sie kodiert werden.

API:

Funktion	Bedeutung
`testfunktion()`	liest String `s` von der Kommandozeile und gibt die Kodierung/Dekodierung von `s` bezgl. Morsecode aus

```python
# morsen.py
#-----------------------------------------------------------------------------------------------------
import sys

# Teil 1: Einlesen des Morsealphabets ------------------------------------------------------------
# uebersetzerErzeugen() liest das Morsealphabet aus der Datei morsealphabet.txt.
# Sie gibt ein Dictionary mit der Zuordnung Buchstabe->Morsezeichen
# und ein Dictionary mit der Zuordung Morsezeichen->Buchstabe zurück.
def uebersetzerErzeugen():
  eingabedatei = open('morsealphabet.txt','r')
  codeDict     = dict()       # Das Dictionary für die Kodierung in den Morsecode.
  decodeDict   = dict()       # Das Dictionary für die Dekodierung aus dem Morsecode.

  for zeile in eingabedatei:  # Gehe durch alle Zeilen der Eingabedatei.
     z = str.split(zeile)       # Teile die Zeile an Folgen von Leerzeichen auf.
     if len(z)<2: continue      # Falls dabei nicht mindestens 2 Teile herauskommen: ignoriere es.
     codeDict[z[0]]   = z[1]    # Trage die Zuordnung Zeichen->Morsezeichen in codeDict ein.
     decodeDict[z[1]] = z[0]    # Trage die Zuordung Morsezeichen->Zeichen in decodeDict ein.

  eingabedatei.close()
  return codeDict, decodeDict
```

```python
# morsen.py
#-----------------------------------------------------------------------------------------------
# Teil 2: Kodieren bzw. Dekodieren eines Wortes ------------------------------------------------

# dekodiere(s, decodeDict) dekodiert den String s zeichenweise aus dem Morsecode.
# Das Dictionary decodeDict hat Morsezeichen als Schlüssel und Buchstaben als Werte.
def dekodiere(s, decodeDict):
  ergebnis = ''
  # Die Morsezeichen sind durch Leerzeichen getrennt und werden der Reihe nach einzeln dekodiert.
  for c in str.split(s):
    ergebnis += decodeDict[c]
  return ergebnis

# kodiere(s, codeDict) kodiert den String s zeichenweise in den Morsecode.
# Das Dictionary codeDict hat Buchstaben als Schlüssel und Morsezeichen als Werte.
def kodiere(s, codeDict):
  ergebnis = ''
  # Die Buchstaben werden der Reihe nach einzeln kodiert.
  for zeichen in s:
    ergebnis += codeDict[zeichen] + ' '
  return ergebnis
```

```python
# morsen.py
#-------------------------------------------------------------------------------------------------
# Die Testfunktion erzeugt die beiden Dictionaries zum Kodieren bzw. Dekodieren,
# und liest ein Wort von der Kommandozeile.
# Wenn das Wort mit . oder - beginnt, dann ist es im Morsecode und wird dekodiert.
# Anderenfalls wird es in den Morsecode kodiert.
# Am Ende wird das (de)kodierte Wort ausgegeben.
def testprogramm():
  codeDict, decodeDict = uebersetzerErzeugen()
  eingabe = sys.argv[1]
  # Falls die Eingabe mit . oder - anfängt, ist sie im Morsecode und wird dekodiert.
  if eingabe[0] in '.-':
      ausgabe = dekodiere(eingabe,decodeDict)
  # Sonst wird die Eingabe in den Morsecode kodiert.
  else:
      ausgabe = kodiere(eingabe,codeDict)

  print(ausgabe)

#-------------------------------------------------------------------------------------------------
if __name__=='__main__': testfunktion()
#-------------------------------------------------------------------------------------------------
```

```python
# morsen.py
#-------------------------------------------------------------------------------------------------
# Die Testfunktion erzeugt die beiden Dictionaries zum Kodieren bzw. Dekodieren,
# und liest ein Wort von der Kommandozeile.
# Wenn das Wort mit . oder - beginnt, dann ist es im Morsecode und wird dekodiert.
# Anderenfalls wird es in den Morsecode kodiert.
# Am Ende wird das (de)kodierte Wort ausgegeben.
def testprogramm():
  codeDict, decodeDict = uebersetzerErzeugen()
  eingabe = sys.argv[1]
  # Falls die Eingabe mit . oder - anfängt, ist sie im M
  if eingabe[0] in '.-':
      ausgabe = dekodiere(eingabe,decodeDict)
  # Sonst wird die Eingabe in den Morsecode kodiert.
  else:
      ausgabe = kodiere(eingabe,codeDict)

  print(ausgabe)

#-------------------------------------------------------------------------------------------------
if __name__=='__main__': testfunktion()
#-------------------------------------------------------------------------------------------------
```

```
mundhenk@ma3: ~/Python/VL11
python@home: python3 morsen.py INFORMATIK
.. -. ..-. --- .-. -- .- - .. -.-
python@home:
```

```python
# morsen.py
#-------------------------------------------------------------------------------------------------
# Die Testfunktion erzeugt die beiden Dictionaries zum Kodieren bzw. Dekodieren,
# und liest ein Wort von der Kommandozeile.
# Wenn das Wort mit . oder - beginnt, dann ist es im Morsecode und wird dekodiert.
# Anderenfalls wird es in den Morsecode kodiert.
# Am Ende wird das (de)kodierte Wort ausgegeben.
def testprogramm():
  codeDict, decodeDict = uebersetzerErzeugen()
  eingabe = sys.argv[1]
  # Falls die Eingabe mit . oder - anfängt, ist sie im Morsecode
  if eingabe[0] in '.-':
      ausgabe = dekodiere(eingabe,decodeDict)
  # Sonst wird die Eingabe in den Morsecode kodiert.
  else:
      ausgabe = kodiere(eingabe,codeDict)

  print(ausgabe)

#-------------------------------------------------------------------------------------------------
if __name__=='__main__': testfunktion()
#-------------------------------------------------------------------------------------------------
```

```
mundhenk@ma3: ~/Python/VL11
python@home: python3 morsen.py INFORMATIK
.. -. ..-. --- .-. -- .- - .. -.-
python@home: python3 morsen.py '.. -. ..-. --- .-. -- .- - .. -.-'
INFORMATIK
python@home:
```

6 Methoden für den Datentyp `str`

Auch die grundlegenden Datentypen in Python sind Klassen – es gibt Methoden für sie.
Wir schauen uns das hier für Strings (unveränderbarer Datentyp `str`) an.

APIs einiger Methoden (es gibt sehr viel mehr):

`str(x)`	gibt ein `str`-Objekt zurück, das aus `x` entsteht `str('text')` wird abkürzend als `'text'` geschrieben
`s[i]`	gibt das `i`-te Zeichen von String `s` zurück; wie bei Arrays beginnt die Zählung mit 0
`s.split()`	gibt ein Array mit den Token von String `s` zurück; ein Token ist eine verbundene Folge von Zeichen ohne Leerzeichen, Tabulator und Zeilenumbruch
`s.strip()`	gibt das `str`-Objekt zurück, das aus String `s` entsteht, indem Leerzeichen, Tabulatoren und Zeilenumbrüche am Anfang und am Ende entfernt werden
`s.isalnum()`	gibt `True` zurück, falls String `s` nur aus alphanumerischen Zeichen besteht
`s.find(t)`	gibt den kleinsten Index (`int`) zurück, an dem String `t` in String `s` beginnt (bzw. -1, falls `t` nicht vorkommt)
`s.replace(alt,neu)`	gibt ein `str`-Objekt zurück, das aus String `s` entsteht, indem jedes Vorkommen des Strings `alt` durch den String `neu` ersetzt wird.

Mehr über Python herausbekommen mit pydoc

Mehr über Python herausbekommen mit pydoc

```
mundhenk@ma3: ~/Python/VL11/NGramme
PYDOC3.4(1)                    General Commands Manual                    PYDOC3.4(1)

NAME
       pydoc3.4 - the Python documentation tool

SYNOPSIS
       pydoc3.4 name

       pydoc3.4 -k keyword

       pydoc3.4 -p port

       pydoc3.4 -g

       pydoc3.4 -w module [...]

DESCRIPTION
       pydoc3.4  name  Show  text  documentation on something.  name may be the name of a Python
       keyword, topic, function, module, or package, or a dotted reference to a class  or  func-
       tion  within  a module or module in a package.  If name contains a '/', it is used as the
       path to a Python source file to document. If name is 'keywords', 'topics', or  'modules',
       a listing of these things is displayed.
Manual page pydoc3(1) line 1 (press h for help or q to quit)
```

Mehr über Python herausbekommen mit pydoc

Mehr über Python herausbekommen mit pydoc

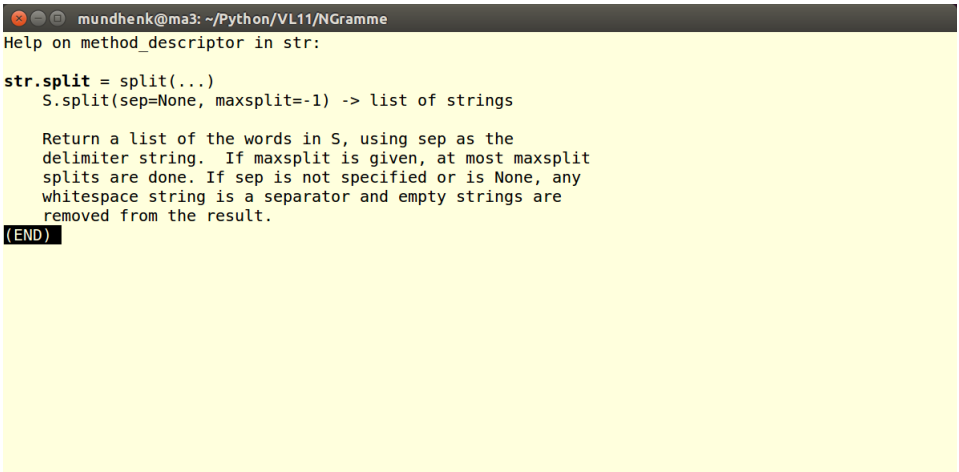
```
mundhenk@ma3: ~/Python/VL11/NGramme
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
:
```

Mehr über Python herausbekommen mit pydoc

```
mundhenk@ma3: ~/Python/VL11/NGramme
|      If chars is given and not None, remove characters in chars instead.
|
|  split(...)
|      S.split(sep=None, maxsplit=-1) -> list of strings
|
|      Return a list of the words in S, using sep as the
|      delimiter string.  If maxsplit is given, at most maxsplit
|      splits are done. If sep is not specified or is None, any
|      whitespace string is a separator and empty strings are
|      removed from the result.
|
|  splitlines(...)
|      S.splitlines([keepends]) -> list of strings
|
|      Return a list of the lines in S, breaking at line boundaries.
|      Line breaks are not included in the resulting list unless keepends
|      is given and true.
|
|  startswith(...)
|      S.startswith(prefix[, start[, end]]) -> bool
|
|      Return True if S starts with the specified prefix, False otherwise.
:
```

Mehr über Python herausbekommen mit pydoc

Mehr über Python herausbekommen mit pydoc



7 Programmierauftrag:
erzeuge einen Text, der ähnlich zu einem vorgegebenen Text ist

Schreiben wie Goethe

1. Statistische Eigenschaft des Ausgangstextes:

zerlege den Ausgangstext in n-Gramme („Abschnitte aus n Zeichen“)
und bestimme für jedes n-Gramm
die Wahrscheinlichkeiten der im Ausgangstext darauf folgenden Buchstaben
(bedingte Wahrscheinlichkeiten, Markov-Modell)

2. Erzeugen eines ähnlichen Textes:

starte mit einem n-Gramm und
hänge ans Ende Buchstaben entsprechend der o.g. Wahrscheinlichkeit für das n-Gramm am Ende des Textes.

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

```
abbb ababb abbabb. abb.
```

hinter ...	kommt im Text ... vor

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

hinter ...	kommt im Text ... vor

a b b b a b a b b a b b a b b . a b b .

Vostellung vom Ablauf des Programmes

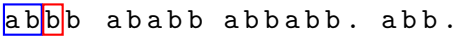
2-Gramm-Analyse eines Textes:

a b b b a b a b b a b b a b b. a b b.

hinter ...	kommt im Text ... vor

Vostellung vom Ablauf des Programmes

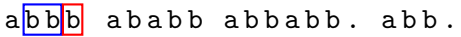
2-Gramm-Analyse eines Textes:



hinter ...	kommt im Text ... vor
ab	b

Vostellung vom Ablauf des Programmes

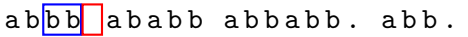
2-Gramm-Analyse eines Textes:



hinter ...	kommt im Text ... vor
ab	b
bb	b

Vostellung vom Ablauf des Programmes

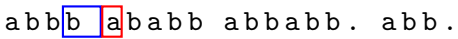
2-Gramm-Analyse eines Textes:



hinter ...	kommt im Text ... vor
ab	b
bb	b ␣

Vostellung vom Ablauf des Programmes

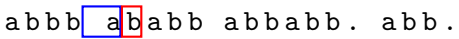
2-Gramm-Analyse eines Textes:



hinter ...	kommt im Text ... vor
ab	b
bb	b ␣
b␣	a

Vostellung vom Ablauf des Programmes

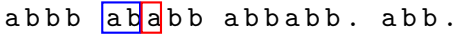
2-Gramm-Analyse eines Textes:



hinter ...	kommt im Text ... vor
ab	b
bb	b ␣
b␣	a
␣a	b

Vostellung vom Ablauf des Programmes

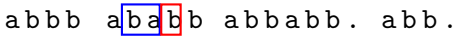
2-Gramm-Analyse eines Textes:



hinter ...	kommt im Text ... vor
ab	b a
bb	b ␣
b␣	a
␣a	b

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:



hinter ...	kommt im Text ... vor
ab	b a
bb	b ␣
b␣	a
␣a	b
ba	b

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

abbb ababb abbabb. abb.

hinter ...	kommt im Text ... vor
ab	b a b
bb	b ␣
b␣	a
␣a	b
ba	b

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

abbb ababb abbabb. abb.

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

```
abbb ababb abbabb. abb.
```

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Erzeugen eines zufälligen Textes:

```
ab
```

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

abbb ababb abbabb. abb.

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Erzeugen eines zufälligen Textes:

a b

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

abbb ababb abbabb. abb.

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Erzeugen eines zufälligen Textes:

abb

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

abbb ababb abbabb. abb.

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Erzeugen eines zufälligen Textes:

a b b

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

abbb ababb abbabb. abb.

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Erzeugen eines zufälligen Textes:

a b b

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

abbb ababb abbabb. abb.

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Erzeugen eines zufälligen Textes:

abb

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

```
abbb ababb abbabb. abb.
```

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Erzeugen eines zufälligen Textes:

```
abb a
```

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

abbb ababb abbabb. abb.

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Erzeugen eines zufälligen Textes:

abb a

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

abbb ababb abbabb. abb.

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Erzeugen eines zufälligen Textes:

abb ab

Vostellung vom Ablauf des Programmes

2-Gramm-Analyse eines Textes:

```
abbb ababb abbabb. abb.
```

hinter ...	kommt im Text ... vor					
ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

Erzeugen eines zufälligen Textes:

```
abb ab
```

Datenstruktur für das Ergebnis der n-Gramm-Analyse:

Dictionary mit *n*-Grammen als Schlüsseln und Arrays aller Folgezeichen als Werte.

Grobe Struktur des Programms und der Daten

Eingabedatei:

```
_Faust._
_Eine Tragödie._
_von_
_Goethe._
```

lies den Eingabetext und bereite ihn für die Analyse vor

```
'Faust. Eine Tragödie. von Goethe. ...'
```

analysiere den Text und erzeuge das *n*-Gramm-Dictionary

```
... 'lebt ': ['i', 'u', 'S', 'U', 'M', 'D', 'D'], ...
```

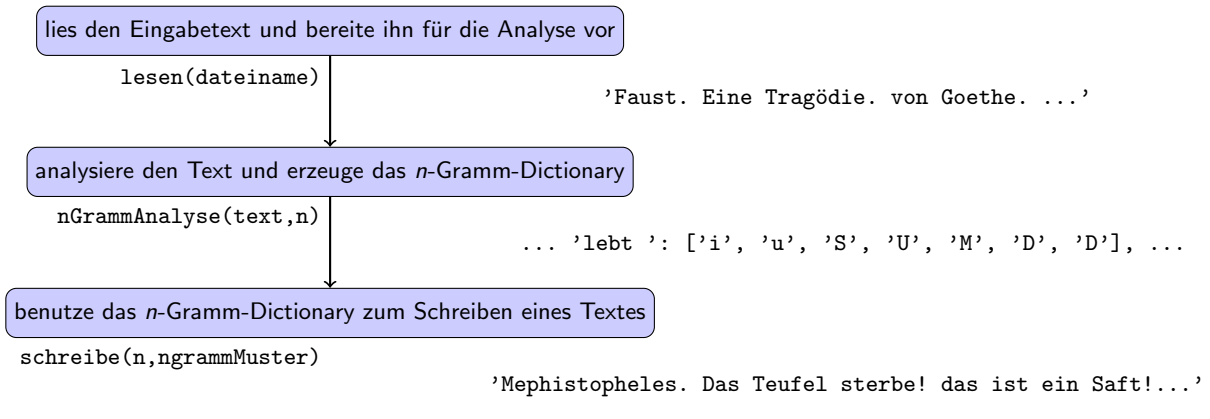
benutze das *n*-Gramm-Dictionary zum Schreiben eines Textes

```
'Mephistopheles. Das Teufel sterbe! das ist ein Saft!...'
```

Grobe Struktur des Programms und der Daten

Eingabedatei:

```
_Faust._
_Eine Tragödie._
_von_
_Goethe._
```



APIs der Funktionen

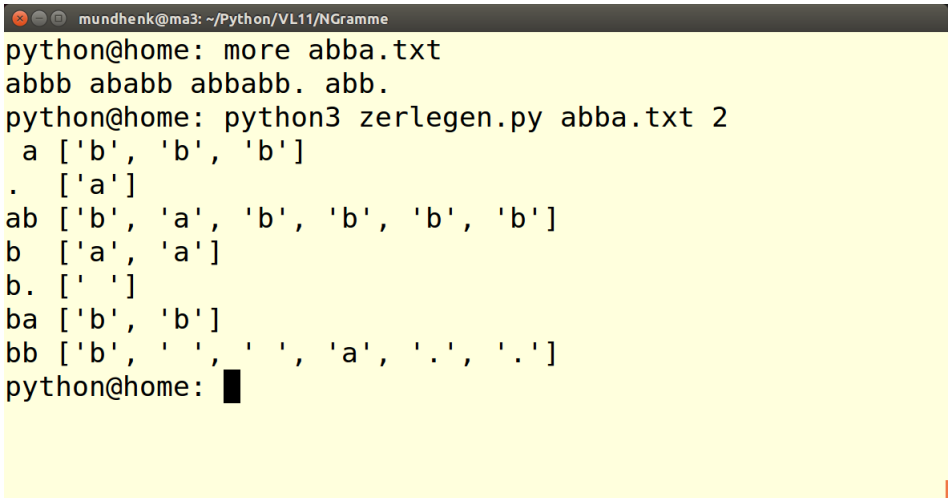
Funktion	Bedeutung
`lesen(dateiname)`	gibt den Text (`str`) aus der Datei `dateiname` (`str`) aus Zeichen, die alphanumerisch oder Satzendezeichen sind, und ohne Folgen von Leerzeichen zurück.
`nGrammAnalyse(text, n)`	gibt das Ergebnis der *n*-Gramm-Analyse (`int n`) von `text` (`str`) als Dictionary zurück. Schlüssel sind die *n*-Gramme von `text` und Werte sind Arrays aus allen Folgebuchstaben des *n*-Gramms (mit Mehrfachvorkommen von Buchstaben).
`schreibe(n, ngrammMuster)`	gibt einen String aus 200 Zeichen zurück, der mit einem zufällig gewählten *n*-Gramm (`int n`) aus `ngrammMuster` (`dict`) beginnt und entsprechend den in `ngrammMuster` enthaltenen Wahrscheinlichkeiten fortgesetzt wird.

Modul einlesen.py mit der Funktion lesen()

```
def lesen(dateiname):
  # Öffne die Datei.
  datei = open(dateiname, 'r')
  # text enthält den aus der Datei übernommenen Text.
  text = ' '
  # Alle Zeilen der Datei werden zeichenweise durchgegangen. '\n' am Zeilenende wird entfernt.
  for zeile in datei:
    for c in zeile.rstrip():
      # Wenn das Zeichen alphanumerisch oder ein Satzendezeichen ist
      # oder wenn es ein Leerzeichen ist, das nicht auf ein Leerzeichen folgt, dann wird es in text übernommen.
      if c.isalnum() or c in '.!?' or (c.isspace() and not text[-1].isspace()): text +=  c
    # Am Zeilenende wird ein Leerzeichen eingefügt, falls nötig.
    if not text[-1].isspace(): text += ' '
  datei.close()
  return text.strip()
#-------------------------------------------------------------------------------------------------------
def test():
  #  Liest einen Dateinamen ein, wandelt die Datei mit lies() in einen String um und gibt den String aus.
  import sys
  dateiname = sys.argv[1]
  text = lesen(dateiname)
  print(text)
#-------------------------------------------------------------------------------------------------------
if __name__ == '__main__': test()
```

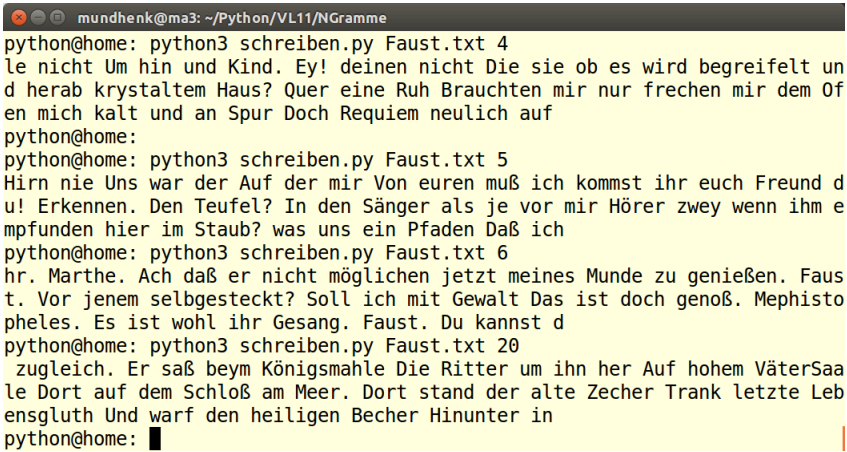
Modul zerlegen.py mit der Funktion nGrammAnalyse()

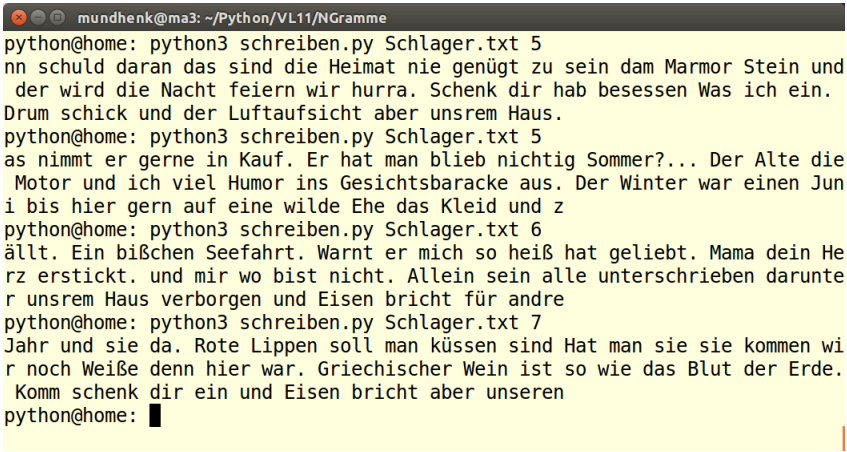
```
def nGrammAnalyse(text, n):
  # ngrammMuster ist das Dictionary aus n-Grammen und Folgebuchstabenarrays
  ngrammMuster = {}
  # Gehe durch alle Stellen des Textes mit einem Folgebuchstaben.
  for i in range(0,len(text)-n):
    ngramm = text[i:i+n]
    naechstesZeichen = text[i+n]
    # Falls noch kein Einintrag für das ngramm im Dictionary ist, erzeuge ihn.
    if ngramm not in ngrammMuster: ngrammMuster[ngramm] = []
    # Trage naechstesZeichen im Dictionary als Folgzeichen von ngramm ein.
    ngrammMuster[ngramm] += [ naechstesZeichen ]
  return ngrammMuster
#-------------------------------------------------------------------------------------------
def test():
  #   Lies einen Dateinamen und n von der Kommandozeile,
  #   erzeuge das Dictionary aus n-Grammen und Folgebuchstaben und gib es aus.
  import sys
  from einlesen import lesen
  text = lesen(sys.argv[1])
  automat = nGrammAnalyse(text, int(sys.argv[2]))

  for k in sorted(automat.keys()):
    print(k, automat[k])
#-------------------------------------------------------------------------------------------
if __name__ == '__main__': test()
```


mundhenk@ma3: ~/Python/VL11/NGramme
python@home: more abba.txt
abbb ababb abbabb. abb.
python@home: python3 zerlegen.py abba.txt 2
 a ['b', 'b', 'b']
. ['a']
ab ['b', 'a', 'b', 'b', 'b', 'b']
b ['a', 'a']
b. [' ']
ba ['b', 'b']
bb ['b', ' ', ' ', 'a', '.', '.']
python@home:

Modul schreiben.py mit der Funktion schreibe()

```
from zerlegen import nGrammAnalyse
from einlesen import lesen
import sys, random
#-------------------------------------------------------------------------------------------------

def schreibe(n, ngrammMuster):
  # Wähle einen Satzanfang.
  satz = random.choice(list(ngrammMuster))
  # Setze den Satz fort, solange es geht und er noch nicht die maximale Länge erreicht hat.
  while satz[-n:] in ngrammMuster and len(satz)<200:
    # Der Satz wird fortgesetzt, indem zufällig ein Folgebuchstabe des ngrammMusters am Satzende angehängt wird.
    satz += random.choice(ngrammMuster[satz[-n:]])
  return satz

#-------------------------------------------------------------------------------------------------
def test():
  #  Liest zwei Argumente von der Kommandozeile: dateiname und n.
  # Es wird ein Satz ausgegeben, der statistisch dem nGrammMuster
  # des Textes in Datei dateiname entspricht.
  text = lesen(sys.argv[1])
  ngrammMuster = nGrammAnalyse(text, int(sys.argv[2]))
  satz = schreibe(int(sys.argv[2]), ngrammMuster)
  print(satz)
#-------------------------------------------------------------------------------------------------
if __name__ == '__main__': test()
```

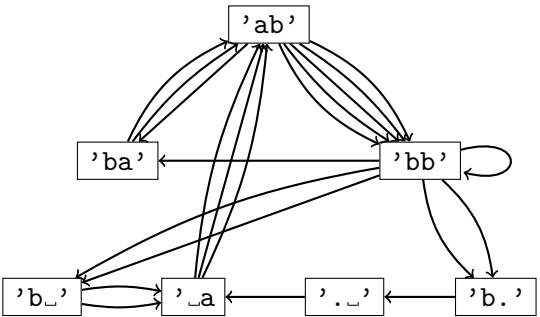

mundhenk@ma3: ~/Python/VL11/NGramme
python@home: python3 schreiben.py Faust.txt 4
le nicht Um hin und Kind. Ey! deinen nicht Die sie ob es wird begreifelt un
d herab krystaltem Haus? Quer eine Ruh Brauchten mir nur frechen mir dem Of
en mich kalt und an Spur Doch Requiem neulich auf
python@home:
python@home: python3 schreiben.py Faust.txt 5
Hirn nie Uns war der Auf der mir Von euren muß ich kommst ihr euch Freund d
u! Erkennen. Den Teufel? In den Sänger als je vor mir Hörer zwey wenn ihm e
mpfunden hier im Staub? was uns ein Pfaden Daß ich
python@home: python3 schreiben.py Faust.txt 6
hr. Marthe. Ach daß er nicht möglichen jetzt meines Munde zu genießen. Faus
t. Vor jenem selbgesteckt? Soll ich mit Gewalt Das ist doch genoß. Mephisto
pheles. Es ist wohl ihr Gesang. Faust. Du kannst d
python@home: python3 schreiben.py Faust.txt 20
zugleich. Er saß beym Königsmahle Die Ritter um ihn her Auf hohem VäterSaa
le Dort auf dem Schloß am Meer. Dort stand der alte Zecher Trank letzte Leb
ensgluth Und warf den heiligen Becher Hinunter in
python@home:


mundhenk@ma3: ~/Python/VL11/NGramme
python@home: python3 schreiben.py Schlager.txt 5
nn schuld daran das sind die Heimat nie genügt zu sein dam Marmor Stein und
der wird die Nacht feiern wir hurra. Schenk dir hab besessen Was ich ein.
Drum schick und der Luftaufsicht aber unsrem Haus.
python@home: python3 schreiben.py Schlager.txt 5
as nimmt er gerne in Kauf. Er hat man blieb nichtig Sommer?... Der Alte die
Motor und ich viel Humor ins Gesichtsbaracke aus. Der Winter war einen Jun
i bis hier gern auf eine wilde Ehe das Kleid und z
python@home: python3 schreiben.py Schlager.txt 6
ällt. Ein bißchen Seefahrt. Warnt er mich so heiß hat geliebt. Mama dein He
rz erstickt. und mir wo bist nicht. Allein sein alle unterschrieben darunte
r unsrem Haus verborgen und Eisen bricht für andre
python@home: python3 schreiben.py Schlager.txt 7
Jahr und sie da. Rote Lippen soll man küssen sind Hat man sie sie kommen wi
r noch Weiße denn hier war. Griechischer Wein ist so wie das Blut der Erde.
Komm schenk dir ein und Eisen bricht aber unseren
python@home:

Eine andere Sicht auf das, was wir gemacht haben

abbb ababb abbabb. abb.

ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					

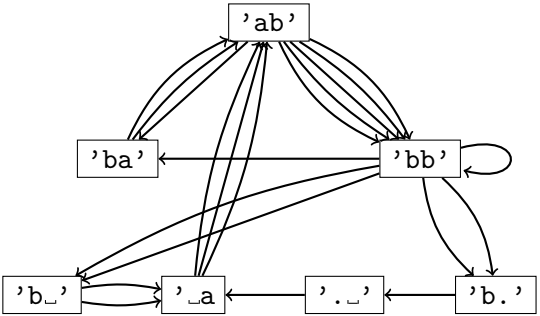


Die n-Gramm-Analyse ergibt ein Netzwerk aus n-Grammen.

Eine andere Sicht auf das, was wir gemacht haben

abbb ababb abbabb. abb.

ab	b	a	b	b	b	b
bb	b	␣	␣	a	.	.
b␣	a	a				
␣a	b	b	b			
ba	b	b				
.␣	a					
b.	␣					



Die n-Gramm-Analyse ergibt ein Netzwerk aus n-Grammen.

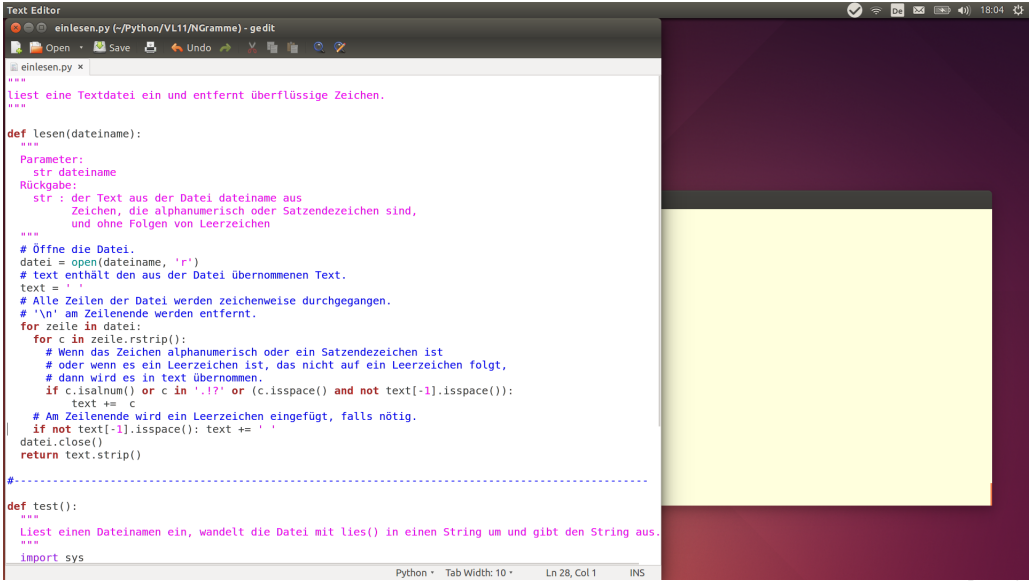
Das Schreiben eines Textes entspricht dem zufälligen Durchlaufen des Netzwerks.

In einem n-Gramm wird jeder ausgehende Link mit gleicher Wahrscheinlichkeit genommen.

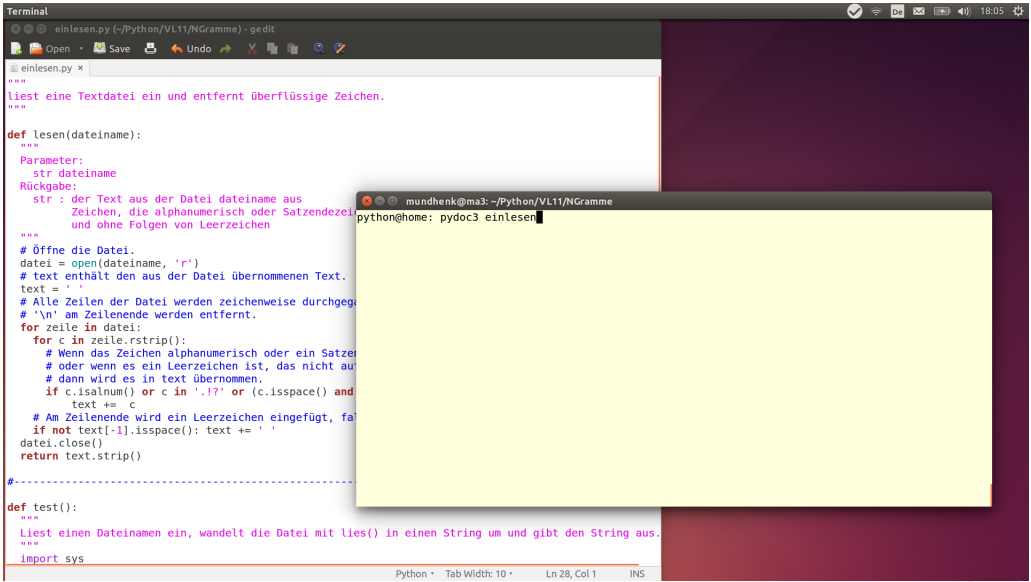
8 Mit Docstrings eigene Module über `pydoc` zugänglich machen

- Der Kommentar ab der ersten Zeile der Datei wird als Beschreibung des Moduls aufgefasst.
- Der Kommentar hinter der Definition einer Funktion (`def fname(...)`) wird als Beschreibung der Funktion aufgefasst.
- `pydoc modulname` (ohne `.py`) zeigt die Beschreibung des Moduls und seiner Funktionen.
- `pydoc modulname.funktionsname` zeigt nur die Beschreibung der Funktion im Modul.
- Damit keine überflüssigen Zeichen angezeigt werden, kann man die Kommentare mit `""" ...Kommentarzeilen... """` einklammern.

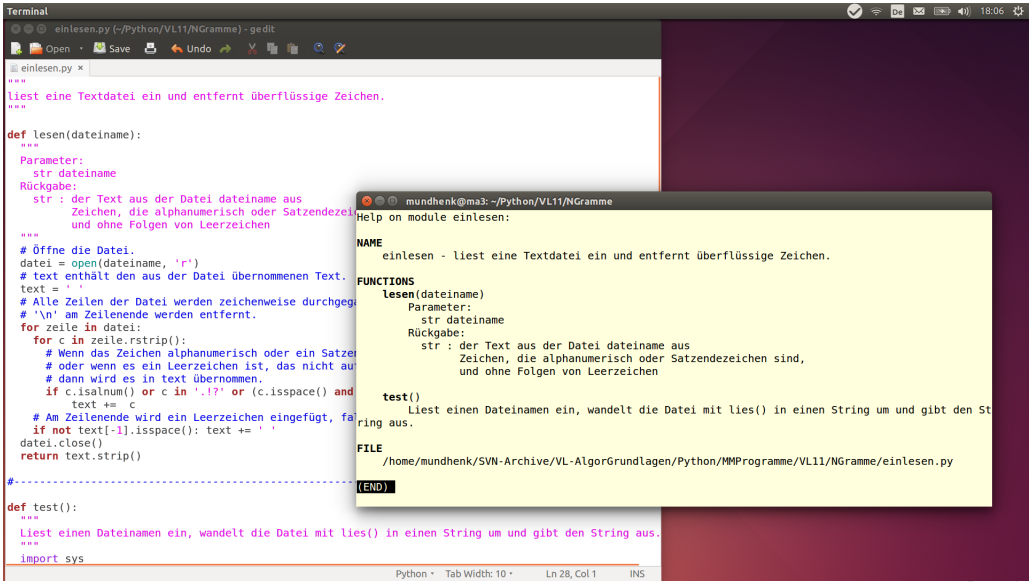
Mit Docstrings eigene Module `pydoc` zugänglich machen



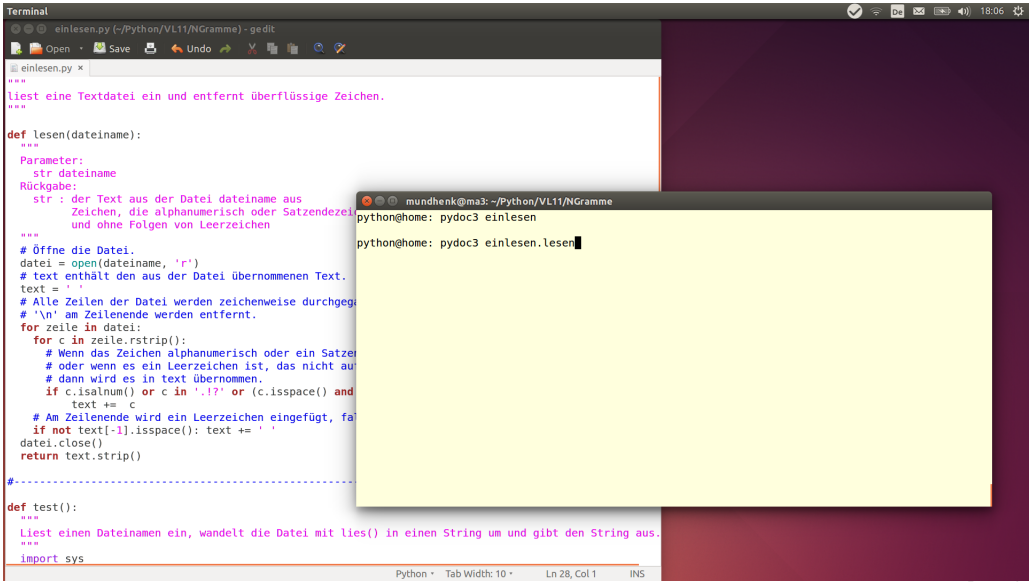
Mit Docstrings eigene Module `pydoc` zugänglich machen



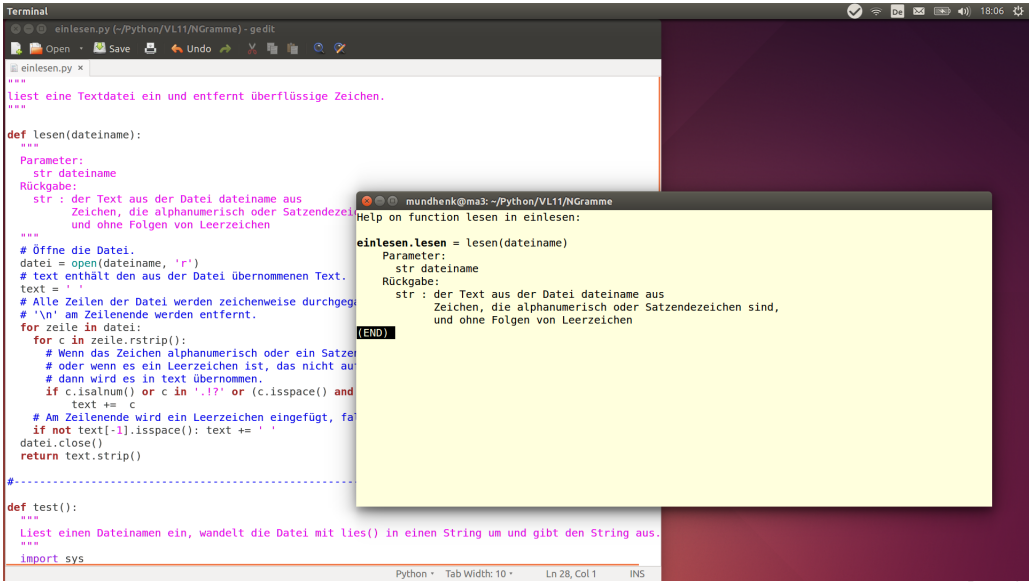
Mit Docstrings eigene Module `pydoc` zugänglich machen



Mit Docstrings eigene Module `pydoc` zugänglich machen



Mit Docstrings eigene Module `pydoc` zugänglich machen



Zusammenfassung

- Dictionaries sind eine unglaublich nützliche Datenstruktur.
  Man kann komplexe Strukturen wie z.B. Netzwerke mit ihnen modellieren.
- Gute Dokumentation der Programme ist wichtig beim Entwickeln größerer Programme.
- Docstrings sind eine gute Möglichkeit, Programme zu dokumentieren.