Unicode in Python, demystifiziert

Marek Kubica

18. September 2008

Das Problem – Internationalisierung

Was heißt das?

Der Fehler

```
UnicodeDecodeError: 'ascii' codec
can't decode byte 0xc4 in position
10: ordinal not in range(128)
```

Was heißt das?

Der Fehler

```
UnicodeDecodeError: 'ascii' codec
can't decode byte 0xc4 in position
10: ordinal not in range(128)
```

- Noch nie diese Exception gesehen?
- Doch gesehen und irgendwie korrigiert?
- Das ist ein *seltsamer* Fehler

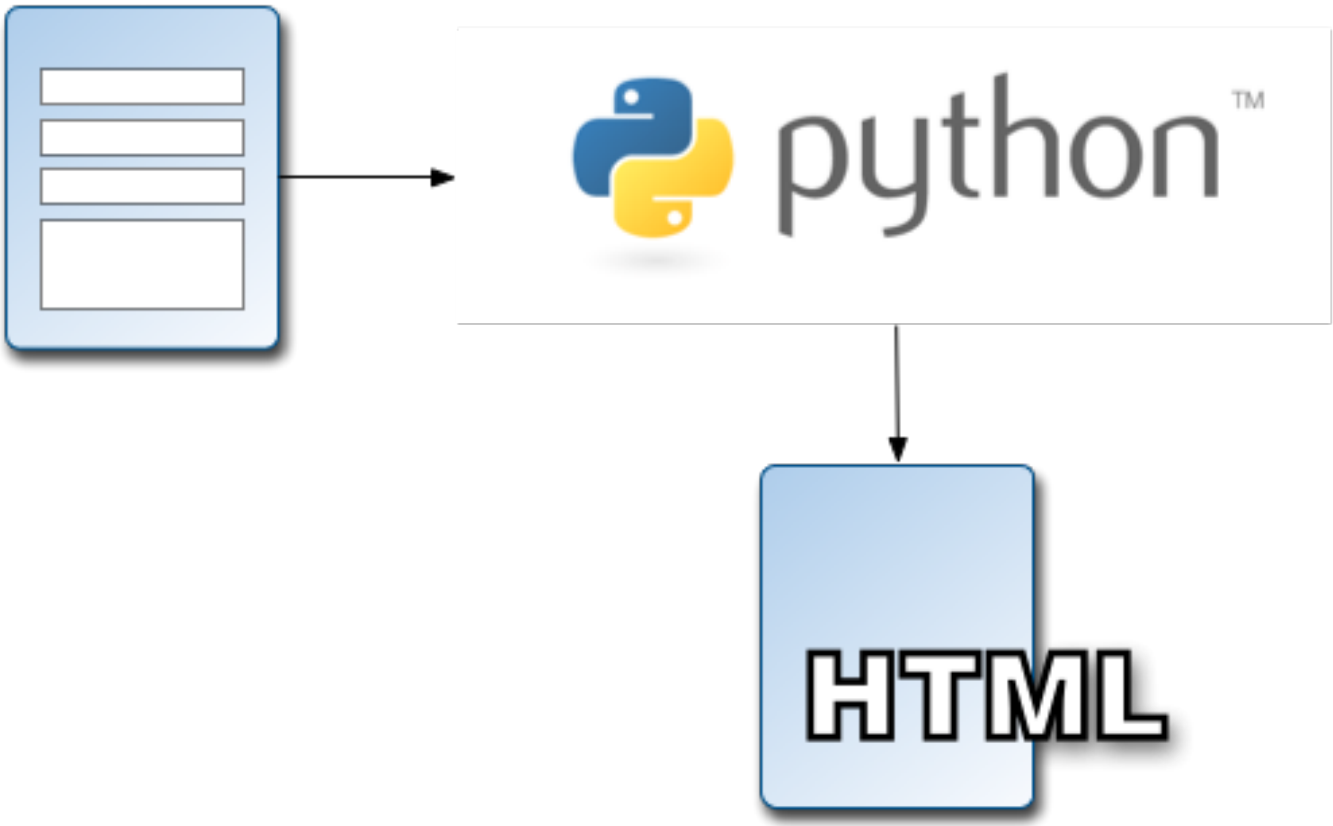
Was gibt es für Anforderungen?

- Sprachen außer Englisch unterstützen
- Fremde Module nutzen
- *beliebige* Texteingaben akzeptieren

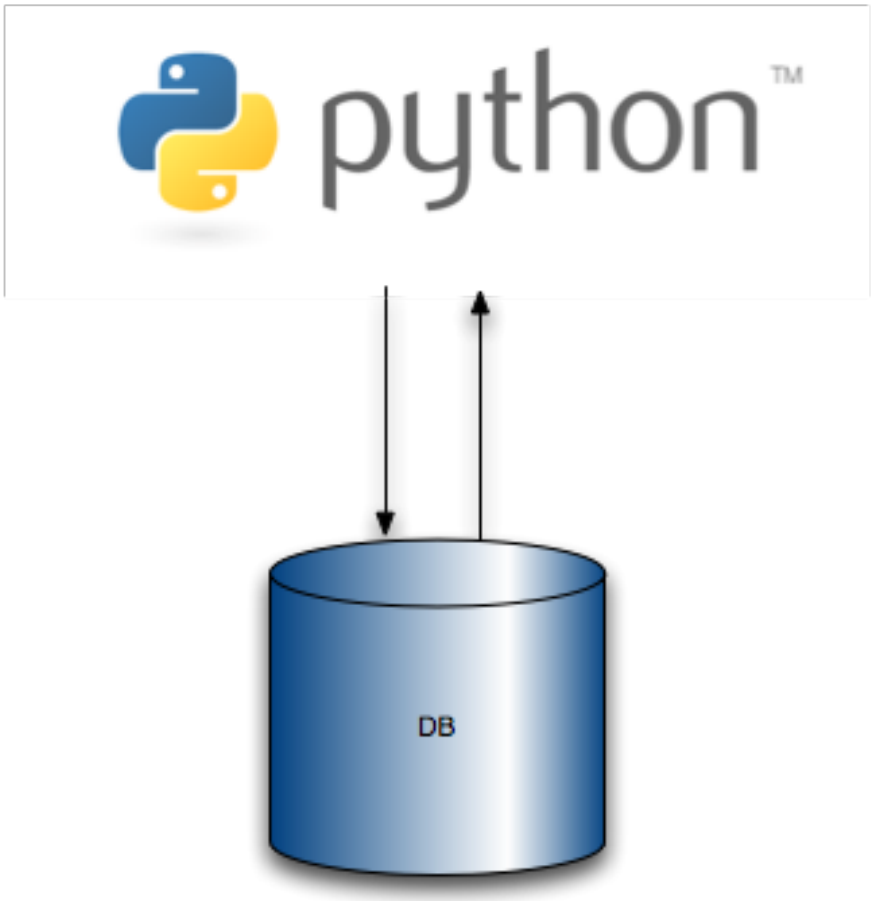
Was gibt es für Anforderungen?

- Sprachen außer Englisch unterstützen
- Fremde Module nutzen
- *beliebige* Texteingaben akzeptieren
  - nie wieder *ue*, *oe* etc.
  - nie wieder Akzente weglassen
  - nie wieder Namen transkribieren
  - nie wieder komische Zeichen auf dem Bildschirm

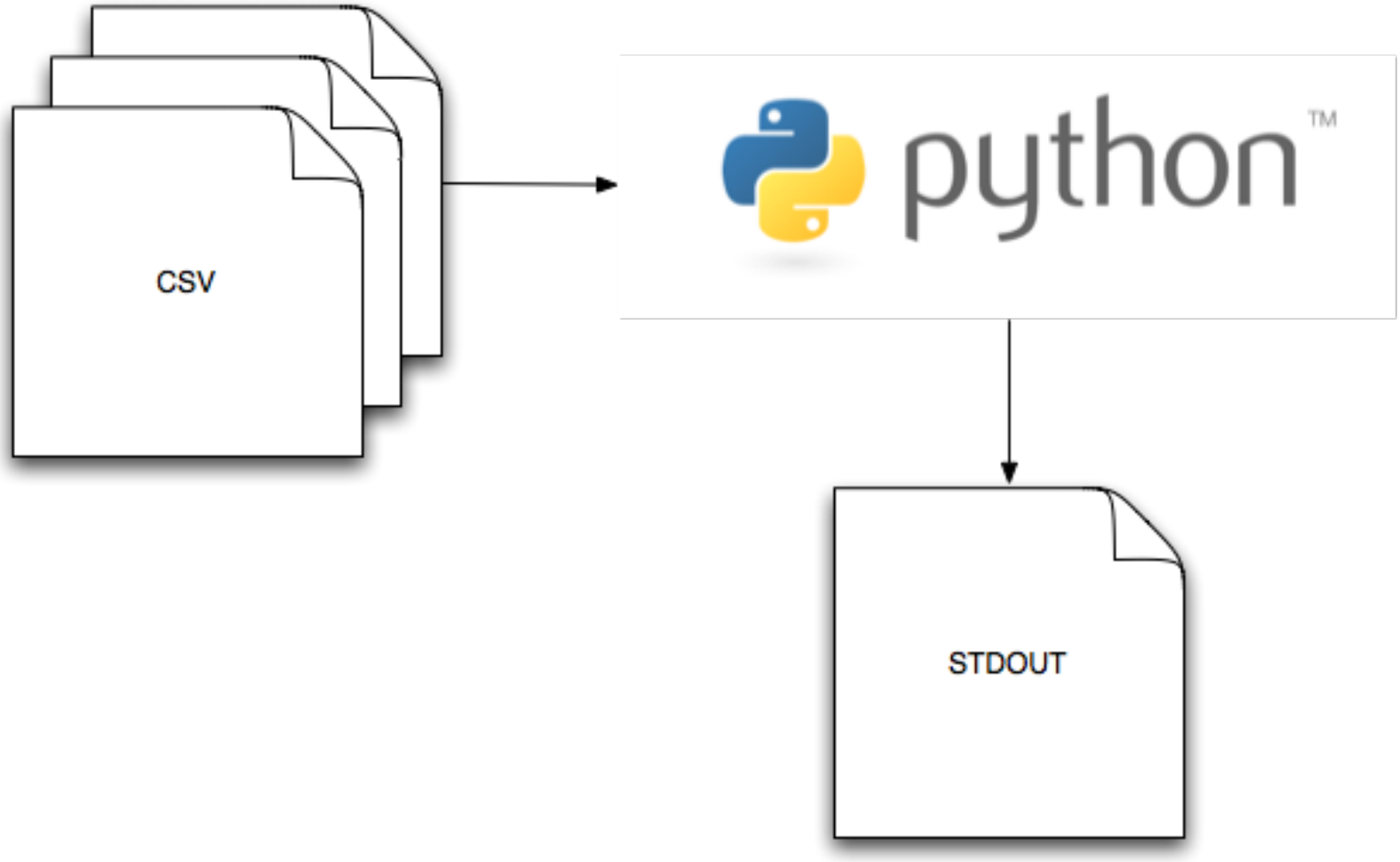
Web-Applikationen



Datenbankinteraktion

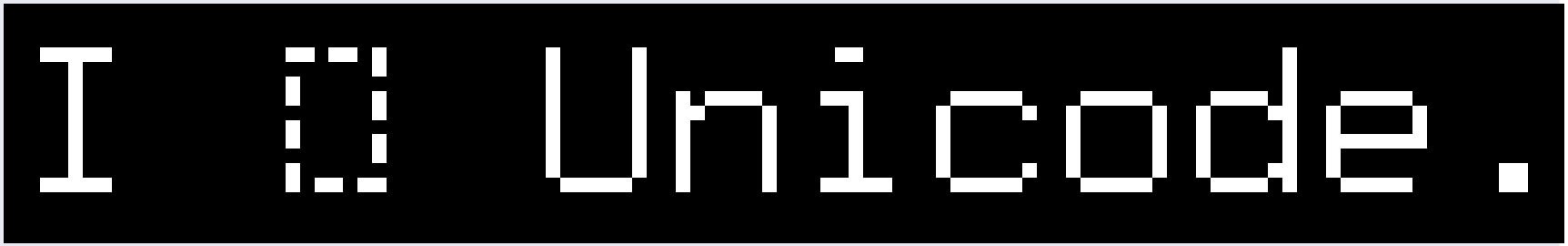


Konsolenprogramm



Die Lösung

Unicode



- An Unicode führt kein Weg vorbei
- Man kann es nicht mehr ignorieren

Unicodekarte

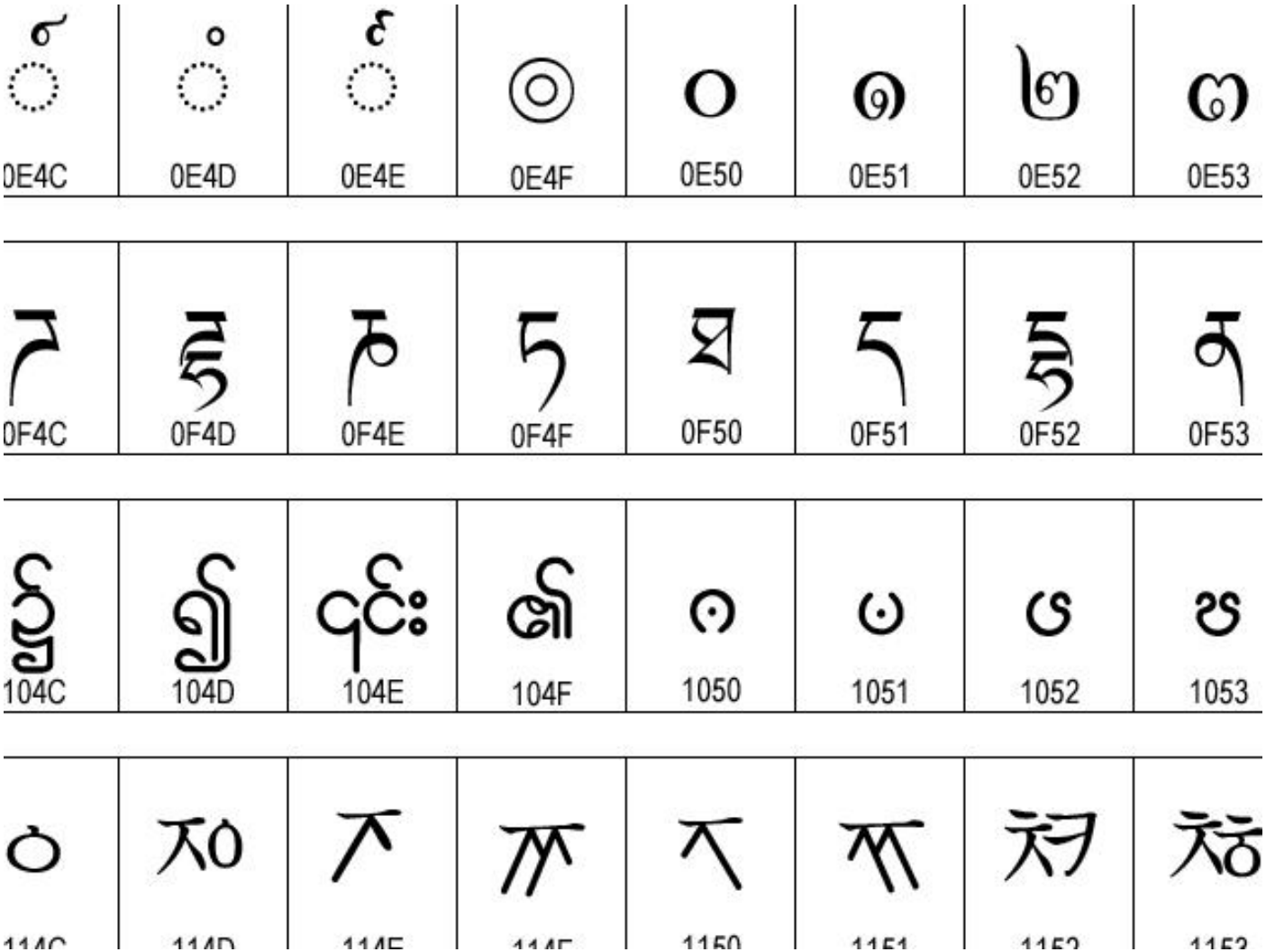
Ian Alberts Unicodekarte

- er hat sich die gesamte Unicodekarte ausgedruckt
- 1.114.112 Codepoints
- 1,8 mal 3,7 Meter
- 22.017 * 42.807 Pixel

Unicodekarte

Unicodekarte 50%

Unicodekarte 100%



Mit Unicode arbeiten

Öffnen wir eine UTF-8 kodierte Datei

Karl Müller

```
>>> f = open('/tmp/karl_utf8.txt', 'r')
>>> karl_utf8 = f.read()
>>> karl_utf8
'Karl M\xc3\xbcller'
```

Was ist es denn?

Im Interpreter

```
>>> karl_utf8
'Karl M\xc3\xbcller'
>>> type(karl_utf8)
<type 'str'>
```

Was ist es denn?

Im Interpreter

```
>>> karl_utf8
'Karl M\xc3\xbcller'
>>> type(karl_utf8)
<type 'str'>
```

- ein String aus Bytes
- 1 Byte = 8 Bits
- ein Bit ist entweder "0" oder "1"

Was ist es denn?

Im Interpreter

```
'Karl M\xc3\xbcller'
```

Was ist es denn?

Im Interpreter

```
'Karl M\xc3\xbcller'
```

- dieser String ist UTF-8-kodiert
- Ein Zeichensatz bezeichnet die Regeln die Zahlen Zeichen (Buchstaben) zuordnet
- Das ü wird durch zwei Bytes repräsentiert
- Andere Zeichensätze können das ü anders darstellen
- Die Python Stdlib unterstützt über 100 Zeichensätze

ASCII

Der wohl bekannteste Zeichensatz

Unser Beispiel

Zeichen	K	a	r	l
Hexadezimal	\x4b	\x61	\x72	\x6c
Dezimal	75	97	114	108

- UTF-8 ist eine Erweiterung von ASCII
- *1963* als "*American* Standard Code for Information Exchange"
- jedes Zeichen ist 1 Byte lang
- nutzt 7 Bit, also $2^7 = 128$ Zeichen möglich

ASCII, fortgesetzt

Der Nachname

Zeichen	M	ü	l	l	e	r
Hexadezimal	\x4d	gibt's nicht	\x6c	\e6c	\x65	\x72
Dezimal	77	gibt's nicht	108	108	101	114

Daraus folgt dann...

ü kann nicht in ASCII kodiert werden

Eingebaute String-Datentypen

Python 2.x

- `<type 'basestring'>`
  - `<type 'str'>`
  - `<type 'unicode'>`

Python 3.0

- nur noch `<type 'str'>`
- verhält sich aber wie `<type 'unicode'>`

Eingebaute String-Datentypen

Python 2.x

- `<type 'basestring'>`
  - `<type 'str'>`
  - `<type 'unicode'>`

Python 3.0

- nur noch `<type 'str'>`
- verhält sich aber wie `<type 'unicode'>`
- `<type 'bytes'>` gibt es nun zusätzlich

Wichtige Methoden

str.decode(encoding)

Von `<type 'str'>` zu `<type 'unicode'>` konvertieren.

unicode.encode(encoding)

Von `<type 'unicode'>` zu `<type 'str'>` konvertieren.

Das Problem

Kann mein Text nicht einfach enkodiert bleiben?

Interpreter

```
>>> karl_utf8
'Karl M\xc3\xbcller'
>>> len(karl_utf8)
12
>>> karl_utf8[7]
'\xbc'
```

Unicode ist unproblematischer

Interpreter

```
>>> karl_utf8
'Karl M\xc3\xbcller'
>>> karl_uni = karl_utf8.decode('utf-8')
>>> karl_uni
u'Karl M\xfcller'
>>> type(karl_uni)
<type 'unicode'>
>>> len(karl_uni)
11
>>> karl_uni[7]
u'\xfc'
```

Über Unicode

Unicode, was soll das sein?

```
u'Karl M\xfcller'
```

- eine Art Text ohne Bytewerte auszudrücken
- eine eindeutige Zahl (Codepoint) für jedes Zeichen jeder Sprache
- unterstützt nahezu alle Sprachen die heutzutage geschrieben werden
- definiert über 1 Million Codepoints

Unicode ist ein Konzept

Buchstabe	Unicode Codepoint
ü	\u00fc = \xfc
€	\u20ac

- Man kann ein Konzept nicht auf Festplatte speichern (abstrakt)
- Man muss es also vorher enkodieren (konkret)

Buchstabe	UTF-8	UTF-16	Latin-1	Latin-9
ü	\xc3\xbc	\xfc\x00	\xfc	\xfc
€	\xe2\x82\xac	\xac\x20	-	\xa4

Notwendig für das Verständnis von Unicode

UTF-8 IST NICHT GLEICH UNICODE

Notwendig für das Verständnis von Unicode

UTF-8 IST NICHT GLEICH UNICODE

- genausowenig wie UTF-16, UTF-32 gleich Unicode sind
- Microsoft nennt UTF-16 Unicode
- IBM verwechselt manchmal UTF mit Unicode (“Unicode encoded”)

Notwendig für das Verständnis von Unicode

UTF-8 IST NICHT GLEICH UNICODE

- genausowenig wie UTF-16, UTF-32 gleich Unicode sind
- Microsoft nennt UTF-16 Unicode
- IBM verwechselt manchmal UTF mit Unicode (“Unicode encoded”)
- dennoch irreführend bis *falsch*

Unicode Transformation Format

```
>>> ab = unicode('AB')
```

UTF-8

```
>>> ab.encode('utf-8')
'AB'
```

- nutzt variable Byteanzahl
- 1 bis 4 Bytes pro Codepoint (*8* bis 32 Bit)
- erste 128 Zeichen identisch mit ASCII

Unicode Transformation Format

```
>>> ab = unicode('AB')
```

UTF-16

```
>>> ab.encode('utf-16')
'\xff\xfeA\x00B\x00'
```

- nutzt variable Byteanzahl
- 2 bis 4 Bytes pro Codepoint (*16* bis 32 Bit)
- für Sprachen deren Zeichen 2 Byte lang sind gut geeignet

Unicode Transformation Format

UTF-32

- feste Byteanzahl, daher schnell zu verarbeiten
- 4 Bytes pro Codepoint (*32* Bit)
- von Python 2.x nicht unterstützt

Text in Unicode dekodieren

- es läuft meist automatisch
- passiert oft in externen Modulen
- Python versucht es zu dekodieren

Python-Magie aufgedeckt

```
>>> karl_uni = u'Karl Müller'
>>> karl_uni
u'Karl M\xfcller'
>>> f = open('/tmp/karl.txt', 'w')
>>> f.write(karl_uni)
Traceback (most recent call last):
  File ''<stdin>'', line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't encode character
u'\xfc' in position 6: ordinal not in range(128)
```

Schluck!

DAS STANDARDENCODING IN PYTHON 2 IST ASCII

Ändere es einfach?!

```
sys.setdefaultencoding('utf-8')
```

Sowas verwenden?

- kann ich das nicht einfach in die `sitecustomize.py` stecken?
- Nein!
- Der Code wird mit anderen Python-Installationen nicht funktionieren
- mehr Aufwand als es wert ist

Lösung

1. Früh in Unicode umwandeln
2. Überall mit Unicode arbeiten
3. So spät wie möglich in Bytestrings konvertieren

1. Früh in Unicode umwandeln

Dekodiere nach <type 'unicode'> so früh wie möglich

```
>>> def to_unicode_or_bust(
...         obj, encoding='utf-8'):
...     if isinstance(obj, basestring):
...         if not isinstance(obj, unicode):
...             obj = unicode(obj, encoding)
...     return obj
...
>>>
```

2. Überall mit Unicode arbeiten

```
>>> to_unicode_or_bust(karl_uni)
u'Karl M\xfcller'
>>> to_unicode_or_bust(karl_utf8)
u'Karl M\xfcller'
>>> to_unicode_or_bust(1234)
1234
```

3. So spät wie möglich in Bytestrings konvertieren

Enkodiere nach <type 'str'> zum speichern auf Festplatte oder zur **Ausgabe**

```
>>> f = open('/tmp/karl_out.txt','w')
>>> f.write(karl_uni.encode('utf-8'))
>>> f.close()
```

Abkürzungen

Lesen

```
>>> import codecs
>>> f = codecs.open('/tmp/karl_utf8.txt', 'r',
...                 encoding='utf-8')
...
>>> f.read()
u'Karl M\xfcller'
>>> f.close()
```

Abkürzungen

Schreiben

```
>>> import codecs
>>> f = codecs.open('/tmp/karl_utf8.txt', 'w',
...                 encoding='utf-8')
...
>>> f.write(karl_uni)
>>> f.close()
```

<module 'codecs'> erspart lästiges Umkodieren

Unicode Inkompatibilität in Python 2

- einige externe Module unterstützen kein Unicode
  - Bugs melden!
- einige Module in der Stdlib unterstützen kein Unicode
  - `csv`

Unicode Workarounds

- in UTF-8 kodieren, danach wieder nach Unicode konvertieren
- Die csv-Dokumentation zeigt wie's geht

```
>>> karl_bytes = karl_uni.encode('utf-8')
>>> # verarbeiten
>>> karl_bytes.decode('utf-8')
u'Karl M\xfcller'
```

Der/die/das BOM

- Byte Order Mark
- manchmal am Anfang der Dateien
- notwendig für Dateien die in UTF-16 und UTF-32 kodiert sind
  - Little Endian
  - Big Endian
- UTF-8 BOM sagt nur “Ich bin UTF-8”
  - populär auf Windows
  - problematisch auf Unix (Shebang)

BOM erkennen

```
>>> f = open('/tmp/karl_utf16.txt','r')
>>> sample = f.read(4)
>>> sample
'\xff\xfeI\x00'
```

- BOM kann 2, 3 oder 4 Bytes lang sein

BOM erkennen

```
>>> import codecs
>>> (sample.startswith(codecs.BOM_UTF16_LE) or
...  sample.startswith(codecs.BOM_UTF16_BE))
...
True
>>> sample.startswith(codecs.BOM_UTF8)
False
```

Muss ich das BOM entfernen

- vielleicht
- UTF-16 dekodieren entfernt das BOM automatisch
- aber nicht UTF-8
  - *es sei denn* man nutzt `s.decode('utf-8-sig')`
    - ab Python 2.5 möglich

Wie errät man das Encoding?

- Es gibt keinen verlässlichen Weg, das Encoding zu erraten
- BOM kann als Hinweis dienen
- `Content-type`-Header enthält üblicherweise `charset=...`
- Das Modul `chardet` versucht es
  - `http://chardet.feedparser.org`
  - geht wie Mozilla (Firefox) vor

Zusammenfassung der Probleme

- das Standardencoding von Python 2 ist 'ascii'
- Dateien können ein BOM enthalten
- nicht alle Python 2-Module in der Stdlib unterstützen Unicode
- Man kann das Encoding nicht zuverlässig erraten

Zusammenfassung der Lösungen

- Enkodiere früh, Unicode überall einsetzen, dekodiere so spät wie möglich
- Wrapper um Libraries schreiben, die kein Unicode können
- Unicode in Unittests
- Wenn man das Encoding rät ist UTF-8 eine gute Wahl
  - Das BOM dient als Indiz
  - `chardet.detect` wenn nichts mehr hilft

Unicode in Python 3

Unicode in Python 3

- Unicode wird nun praktikabel!
- `<type 'str'>` ist ein Unicode-Objekt
- neuer `<type 'bytes'>`-Typ
- alle Module der Stdlib unterstützen Unicode
- keine u"text"-Syntax mehr, dafür b"bytes"
  - 2.6: `from __future__ import unicode_literals`
- `open` nimmt ein Encoding-Argument, wie `codecs.open`
- Standardencoding nun UTF-8 statt ASCII

Weiterführende Informationen

Hauptquelle dieser Folien

Besten Dank an Kumar McMillan für
`http://farmdev.com/talks/unicode/`.

Wiki

- `http://wiki.python.de/Von_Umlauten,_Unicode_und_Encodings`
- `http://wiki.python.de/Unicode`

Weitere Artikel

- `http://www.joelonsoftware.com/articles/Unicode.html`

Das wars

Was jetzt?

Fragen?