PROKRASTINATION, PANIK, DOKU PAUKEN

Was hilft beim Umstieg auf Python 3?

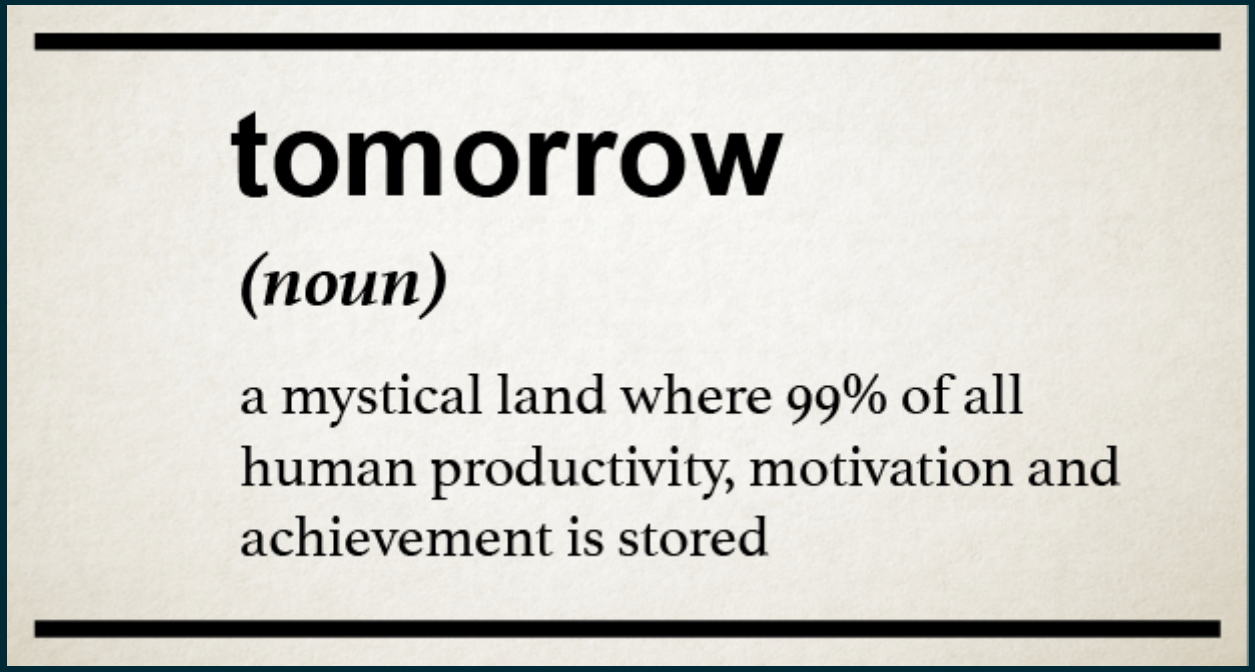
Knut Franke

2018-06-09

PYTHON DREI?

- Python 3 veröffentlicht Ende 2008
- Python 2 Support bis Anfang 2020

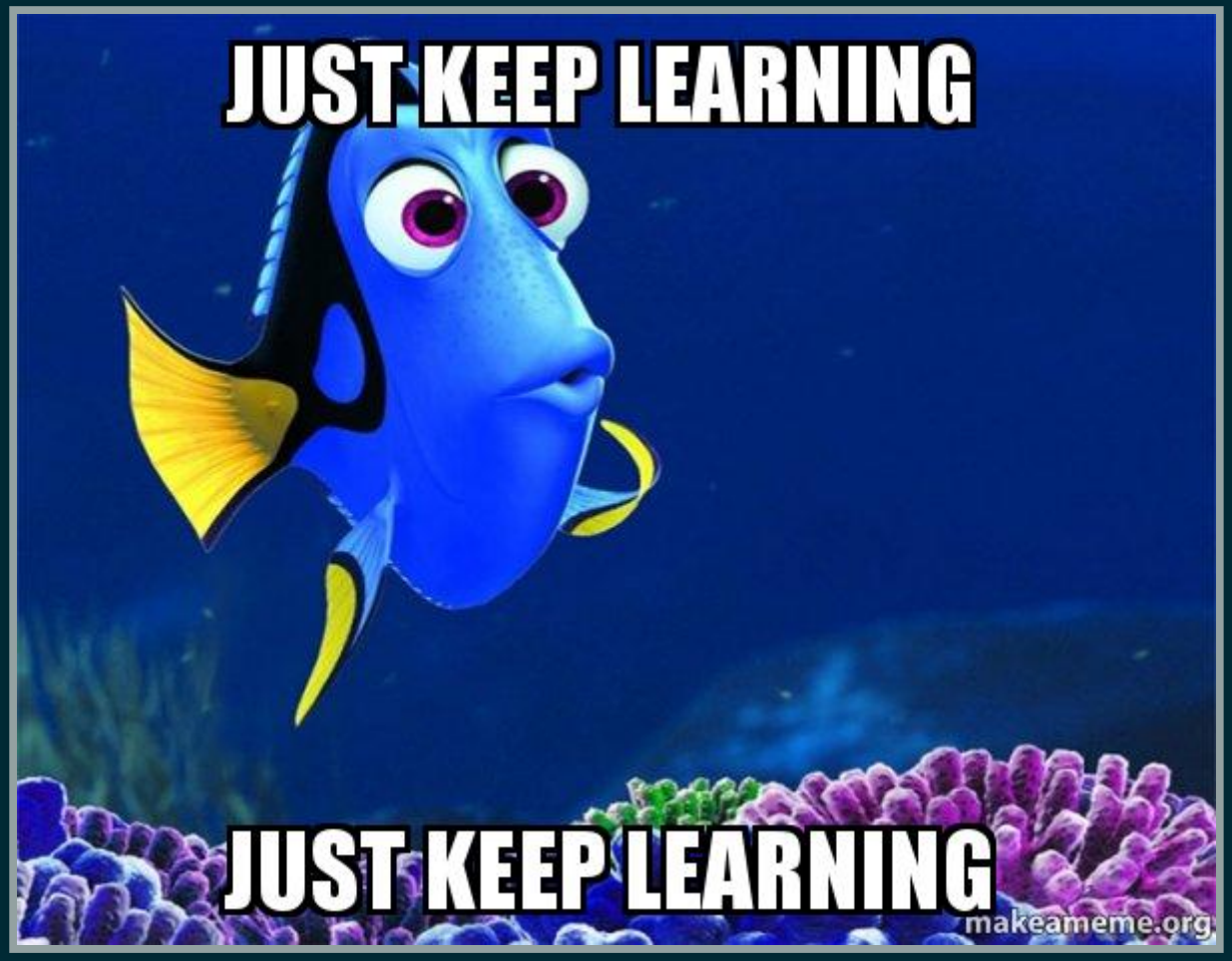
STRATEGIE 1: AUFSCHIEBEN



STRATEGIE 2: PANIK

Panik!

STRATEGIE 3: LERNEN



ERSTE HILFE

- alter Code -> `#!/usr/bin/python2`
- neuer Code -> `#!/usr/bin/python3`
- es sei denn der Code läuft mit beiden Versionen
- Details: PEP 394

FÜR NEUEN CODE...

UNTERSCHIEDE (1)

ALT

```
print 'Hello', 'Guide'
```

NEU

```
from __future__ import print_function
print('Hello', 'Guido')
```

UNTERSCHIEDE (2)

ALT

```
raise ValueError, "dodgy value"
```

NEU

```
raise ValueError("dodgy value")
```

UNTERSCHIEDE (3)

ALT

```
try:
  a = int("invalid")
except ValueError, e:
  ...
```

NEU

```
try:
  a = int("invalid")
except ValueError as e:
   ...
```

UNTERSCHIEDE (4)

ALT

```
assert 2 / 3 == 0
```

NEU

```
assert 2 // 3 == 0
```

UNTERSCHIEDE (5)

- ALT: `0644` NEU: `0o644`
- ALT: `for key in mydict.iterkeys():` NEU: `for key in mydict`
- ALT: `for key, value in mydict.iteritems():` NEU: `for key, value in mydict.items():`
- NEU: `from functools import reduce`

TEXT VS. BYTES

- Python 2 **`unicode`** entspricht Python 3 **`str`**
- Python 2 **`str`** entspricht Python 3 **`bytes`**
- keine implizite Konvertierung in Python 3
- Achtung: Default bei `encode()`/`decode()` in Python 2 ist `ascii`, in Python3 `utf-8` (unabhängig von der OS Einstellung!)
- das `subprocess` Modul liefert stdout/stderr als `bytes`

ENCODINGS

- `sys.getdefaultencoding()` ist selten hilfreich, da versionsabhängig - `ascii` in Python 2 und `utf-8` in Python 3
- Linux: `locale.getpreferredencoding()` (abhängig von `$LC_ALL`)

ENCODINGS (CONTINUED)

- Windows:
  - `sys.getfilesystemencoding()` für Dateinamen, Kommandozeilenargumente und Umgebungsvariablen
  - `locale.getpreferredencoding()` für Dateiinhalte
  - `sys.stderr.encoding` für Terminal-Encoding (Subprozesse)

ENCODINGS (CONTINUED)

```
import sys
import subprocess
p = subprocess.Popen(["date", "/?"], stdout=subprocess.PIPE)
(out, err) = p.communicate()
out_string = out.decode(sys.stdin.encoding)
```

ENCODINGS (I/O)

Um Dateien zu lesen/schreiben, deren Encoding vom Betriebssystem-Encoding abweicht:

```
from io import open # für Python 2
f = open("myutf8file.txt", "w", encoding="utf-8")
```

LISTEN VS. GENERATOREN

- Vorsicht bei **`range()`**, **`map()`**, **`zip()`**, **`filter()`**, **`mydict.keys()`**, **`mydict.values()`**, **`mydict.items()`**
- in Python 2 geben diese eine Liste (d.h. unabhängige Kopie) zurück, in Python 3 dagegen einen Generator
- wird die darunter liegende Datenstruktur verändert, so muss man ggf. die genannten Funktionen noch in **`list(...)`** einschließen.

LISTEN VS. GENERATOREN (BEISPIEL)

```
# richtig
a = list("tuebix")
for c, i in list(zip(a, range(6))):
    if i % 2 == 0:
        a.remove(c)
a # => ['u', 'b', 'x']

# FALSCH
a = list("tuebix")
for c, i in zip(a, range(6)):
    if i % 2 == 0:
        a.remove(c)
a # => ['u', 'e', 'i', 'x']
```

CODE-FORMATIERUNG

- **Spaces** statt Tabs **für Einrückung** benutzen!
- Mischung ist in Python 3 nicht mehr erlaubt

EXISTIERENDEN CODE PORTIEREN

YOU'RE USING VERSION CONTROL, RIGHT?

PORTIERUNG AUF PYTHON 3.X ONLY

Wenn man Python 2.x nicht parallel weiter unterstützen muss, kann man das mit Python 3 mitgelieferte Tool **`2to3`** benutzen:

```
find src -name '**.py' -o -name '**.in' \
| xargs 2to3 --write --nobackups
```

UNTERSTÜTZUNG VON 2.X UND 3.X (AKA 2*3)

- six Paket: https://pythonhosted.org/six
- Empfehlung: nur Versionen 2.6/2.7 und 3.3+ benutzen
  - 2.5 fehlt `print_function`, `unicode_literals` und `io.open()`
  - `3.0 - 3.2` kennen die `u"unicode string"` Syntax nicht.

PYTHON-MODERNIZE

```
pip install modernize
find src -name '*.py' -o -name '*.in' \
| xargs python-modernize --write --nobackups
```

- erzeugter Code benötigt i.d.R. Python >= 2.6

Alternativen:

- python-future/futurize
- sixer
- 2to6

STOLPERSTEINE

- das `subprocess` Modul akzeptiert beim `args` Argument `str`, öffnet `std{in,out,err}` aber standardmäßig als Binärstreams
- verändertes Timing kann latente IPC/threading Probleme zum Vorschein bringen
- `print(..., end=" ")` macht im Gegensatz zu `print ...,` standarmäßig keinen Flush des Ausgabepuffers, daher muss man ggf. explizit `sys.stdout.flush()` aufrufen

STOLPERSTEINE 2

- Doppelter `decode()` eines Bytestrings (`b"abc".decode("utf-8").decode("utf-8")`) ist zwar grundsätzlich ein konzeptioneller Fehler, "funktioniert" mit Python 2 aber solange dabei keine ungültigen Bytefolgen entstehen. In Python 3 bekommt man grundsätzlich einen `AttributeError`.

HAPPY CODING!