PSPP-Praktikum 13
Python (Teil 2)

1. Pipeline in Python *(Abgabe)*

Das folgende Stück Python-Code nimmt an einer Liste von Band-Beschreibungen (in Form von Dictionaries) ein paar Änderungen vor:

```
bands = [{'name': 'sunset rubdown', 'country': 'UK', 'active': False},
         {'name': 'women', 'country': 'Germany', 'active': False},
         {'name': 'a silver mt. zion', 'country': 'Spain', 'active': True}]

def format_bands(bands):
    for band in bands:
        band['country'] = 'Canada'
        band['name'] = band['name'].replace('.', '')
        band['name'] = band['name'].title()

>>> format_bands(bands)
>>> print(bands)
[{'name': 'Sunset Rubdown', 'active': False, 'country': 'Canada'},
 {'name': 'Women', 'active': False, 'country': 'Canada' },
 {'name': 'A Silver Mt Zion', 'active': True, 'country': 'Canada'}]
```

Dieser Programmausschnitt hat noch verschiedene Defizite:

- Wozu dient die Funktion *format_bands*? Kann man hier von einer genau definierten Zuständigkeit sprechen?
- Es gibt kaum wiederverwendbare Teile in diesem Programm, das Programm ist schwer zu testen und auch schwierig zu parallelisieren.

Hier ein stärker funktionaler Ansatz:

```
pipeline_each(bands, [set_canada_as_country,
                      strip_punctuation_from_name,
                      capitalize_names])
```

In diesem Fall wird eine Liste von für alle Bands aufzurufenden Funktionen übergeben. Die Ausgabe der einen Funktion wird jeweils als Eingabe der nächsten Funktion verwendet. Die Funktionen dürfen die Datenstruktur dabei nicht direkt verändern, sondern jeweils eine neue Datenstruktur zurückgeben. Dazu verwenden wir eine Funktion *assoc* (nächste Seite).

Aufgabe: Schreiben Sie die Funktionen *set_canada_as_country*, *strip_punctuation_from_name* und *capitalize_names* mit Hilfe von *assoc*.

```
def assoc(_d, key, value):
    from copy import deepcopy
    d = deepcopy(_d)
    d[key] = value
    return d
```

Da *assoc* eine Kopie vom Dictionary anlegt, verändern diese Funktionen die Ausgangsdatenstruktur nicht. Sie geben eine neue Datenstruktur zurück. Die String-Methoden *replace* und *title* können ebenfalls keinen Schaden anrichten, da Strings in Python unveränderlich sind. In Python ist es wichtig, jeweils zwischen veränderbaren (z.B. Listen, Dictionaries) und nicht veränderbaren (z.B. Tupel, Strings) Datenstrukturen zu unterscheiden. In einer funktionalen Sprache wie Clojure ist dies kaum ein Problem, da unveränderliche Datenstrukturen verwendet werden.

Was noch fehlt, ist die Funktion *pipeline_each*:

```
def pipeline_each(data, fns):
    import functools
    return functools.reduce(lambda a, fn: list(map(fn, a)), fns, data)
```

Aufgabe: Machen Sie sich klar, wie *reduce* und *map* hier zusammenspielen, um die Funktionen der Reihe nach auszuführen.

Ein weiterer Abstraktionsschritt könnte sein, das Gemeinsame aus den drei verwendeten Funktionen *set_canada_as_country*, *strip_punctuation_from_name* und *capitalize_names* zu extrahieren: Es wird ja jeweils ein Attribut in einem Dictionary auf eine bestimmte Art und Weise verändert. Dazu könnte die Funktion *call* dienen:

```
def call(fn, key):
    def apply_fn(record):
        return assoc(record, key, fn(record.get(key)))
    return apply_fn
```

Da Lambda-Ausdrücke in Python sehr eingeschränkt sind, definieren wir eine innere Funktion, die dann mit *return* zurückgegeben wird. Die Funktion *set_canada_as_country* kann dann vereinfacht so geschrieben werden:

```
set_canada_as_country = call(lambda x: 'Canada', 'country')
```

Aufgabe: Drücken Sie die auch die beiden anderen Funktionen mit Hilfe von *call* aus.

Hinweis zur Abgabe:
Geben Sie die Python-Datei mit Ihren Lösungen bis zum nächsten PSPP ab:
https://radar.zhaw.ch/python/UploadAndCheck.html

Praktikum: `PS13`
Name und Kurznamen angeben und *Upload* wählen.

2. Pipeline in Python *(fakultativ)*

(Ergänzung)

Diese Aufgabe schliesst an Aufgabe 1 an.

Eine neue Anforderung soll umgesetzt werden: Für jede Band sind nur noch die beiden Attribute *name* und *country* zurückzugeben. Es zeigt sich, dass dazu nur ein Schritt zur Pipeline hinzugefügt werden muss:

```
>>> pipeline_each(bands, [set_canada_as_country,
                          strip_punctuation_from_name,
                          capitalize_names,
                          select(['name', 'country']) ])
[{'name': 'Sunset Rubdown', 'country': 'Canada'},
 {'name': 'Women', 'country': 'Canada'},
 {'name': 'A Silver Mt Zion', 'country': 'Canada'}]
```

Es zeigt sich, dass die Pipeline einfach erweiterbar ist.

Aufgabe: Implementieren Sie die Funktion *select*. Sie erhält in einem Parameter *keys* eine Liste von Schlüsseln und liefert eine Funktion zurück.

Tipps:

- Auch hier ist es sinnvoll, eine innere Funktion zu definieren, welche Sie dann mit *return* zurückgegeben wird (vergleichbar mit der Funktion *call* oben).
- Es gibt verschiedene Lösungsmöglichkeiten. Wenn Sie *reduce* verwenden, arbeiten Sie die Liste der *keys* ab, beginnen mit einem leeren Dictionary { } als Initialwert und setzen die einzelnen Attribute mit *assoc* (s.o.).

Quellenangaben:

Michael Fogus: Functional JavaScript, Introducing Functional Programming with Underscore.js, O'Reilly, 2013

Mary Rose Cook: A practical introduction to functional programming

2. Python Standard Library *(fakultativ)*

(Ergänzung zu alter Aufgabe)

Im letzten Praktikum wurde anhand eines Beispiels demonstriert, dass mit Hilfe der Python-Standardbibliothek kompakte Lösungen möglich sind. Das Beispiel lädt eine Webseite via HTTP und liefert eine Wortliste des Inhalts. Eine gute Basis, um beispielsweise eine Volltextsuche zu implementieren.

```
import urllib.request
import html
import re

def doload(url):
    contents = urllib.request.urlopen(url).read()
    contents = contents.decode(encoding="utf-8")
    data = html.unescape(contents)
    withouttags = re.sub(r"<(.|\s)*?>", " ", data)
    return re.sub(r"\s+", " ", withouttags).split()
```

Das Modul kann folgendermassen getestet werden:

```
>>> import testthis
>>> testthis.doload("http://dublin.zhaw.ch/~bkrt/hallo.html")
>>> testthis.doload("http://loremipsum.net")
```

Sicher ist Ihnen aufgefallen, dass es sich hier um eine typisch imperative Lösung handelt. Zwischenwerte werden in Variablen gespeichert. Tatsächlich könnten die Lösungsschritte mit entsprechend angepassten und curryfizierten Funktionen eleganter mit Hilfe einer Pipeline umgesetzt werden. Dann könnte es etwa so aussehen:

```
def doload(url):
    pipeline(
        httpGetResource,
        decodeByteStream,
        unescapeData,
        removeTags,
        unifyWhitespaceTo(" "),
        split)(url)
```