PSPP-Praktikum 9
Common Lisp (Teil 2)

1. Weitere Abstraktionen

Im letzten Praktikum haben Sie verschiedene, gleich aufgebaute Funktionen zu einer Funktion *map-list* verallgemeinert (welche in Common Lisp bereits unter dem Namen *mapcar* vorhanden ist). Ausserdem haben Sie Funktionen geschrieben, welche die Elemente einer Liste kombinieren, zum Beispiel addieren oder multiplizieren. Wahrscheinlich sehen Ihre Lösungen etwa so aus:

```
(defun list-sum (seq)
    (cond ((null seq) 0)
          (t (+ (car seq) (list-sum (cdr seq))))))

(defun list-mult (seq)
    (cond ((null seq) 1)
          (t (* (car seq) (list-mult (cdr seq))))))
```

Diese sollen nun ebenfalls verallgemeinert werden.

Aufgabe: Studieren Sie die beiden Funktionen und schreiben Sie eine neue Funktion *reduce-list*, eine verallgemeinerte Version von *list-sum*, *list-mult* und ähnlichen Funktionen. Sie soll drei Parameter haben: eine Funktion, einen Initialwert und die zu verarbeitende Liste.

Tatsächlich existiert auch diese Funktion in Common Lisp bereits. Sie heisst *reduce* und unterstützt diverse Schlüsselwort-Parameter.

Eine weitere gut für die Verarbeitung von Listen geeignete Funktion ist:

```
(defun func (f seq)
  (cond ((null seq) nil)
        ((funcall f (car seq))
            (cons (car seq) (func f (cdr seq))))
        (t (func f (cdr seq)))))
```

Aufgabe: Finden Sie heraus, was diese Funktion macht. Geben Sie ihr einen besseren Namen und testen Sie, ob die Funktion die erwarteten Ergebnisse liefert.

2. Listen erzeugen

In den bisherigen Beispielen haben wir vor allem Listen verarbeitet. Nun soll eine Liste von Zahlen gemäss einer in Form von einem bis drei Argumenten übergebenen Spezifikation erzeugt werden.

Schreiben Sie eine Funktion *range*, die eine Sequenz von Zahlen aus einem bestimmten Bereich erzeugt. Wenn der Funktion nur ein Argument übergeben wird, soll die Zahlenfolge von 0 bis zu dieser Zahl (exklusive) ausgegeben werden. Bei zwei Argumenten stellt das erste Argument die Ausgangszahl (von, inklusive) und das zweite das Ziel (bis, exklusive) dar. In einem dritten Argument kann die Schrittweite angegeben werden. Einige Beispiele sollen das verdeutlichen:

```
> (range 8)
(0 1 2 3 4 5 6 7)

> (range 5 10)
(5 6 7 8 9)

> (range 5 20 3)
(5 8 11 14 17)

> (range 10 2 -1)
(10 9 8 7 6 5 4 3)
```

```
> (range 5 5)
NIL

> (range 5 2)
NIL

> (range -5 0)
(-5 -4 -3 -2 -1)

> (range -5)
NIL
```

Das ist nicht ganz einfach. Überlegen Sie erst, wie Sie ein einfacheres Problem lösen könnten: eine Funktion *range-simple*, die immer zwei Parameter hat, *von* und *bis*. Ist eine rekursive Lösung möglich? Wenn Sie nicht weiterkommen, können Sie sich am Lösungshinweis in der Fussnote[1] orientieren.

Die Funktion *range-simple* können Sie dann zur flexibleren Variante verallgemeinern. In den Code-Beispielen zum Praktikum hat es einige Testfälle. Falls *range* nicht richtig zum Laufen kommt geben Sie *range-simple* ab.

Programmdatei und Abgabe

Stellen Sie die Funktion *range* und allenfalls nötige Hilfsfunktionen aus Aufgabe 2 mit Kommentaren in einer Datei (Erweiterung: *.lisp*) zusammen und geben Sie diese Datei bis zum Beginn des nächsten PSPP-Praktikums ab: https://radar.zhaw.ch/python/UploadAndCheck.html

Praktikum: PS9
Name und Kurznamen angeben und *Upload* wählen.

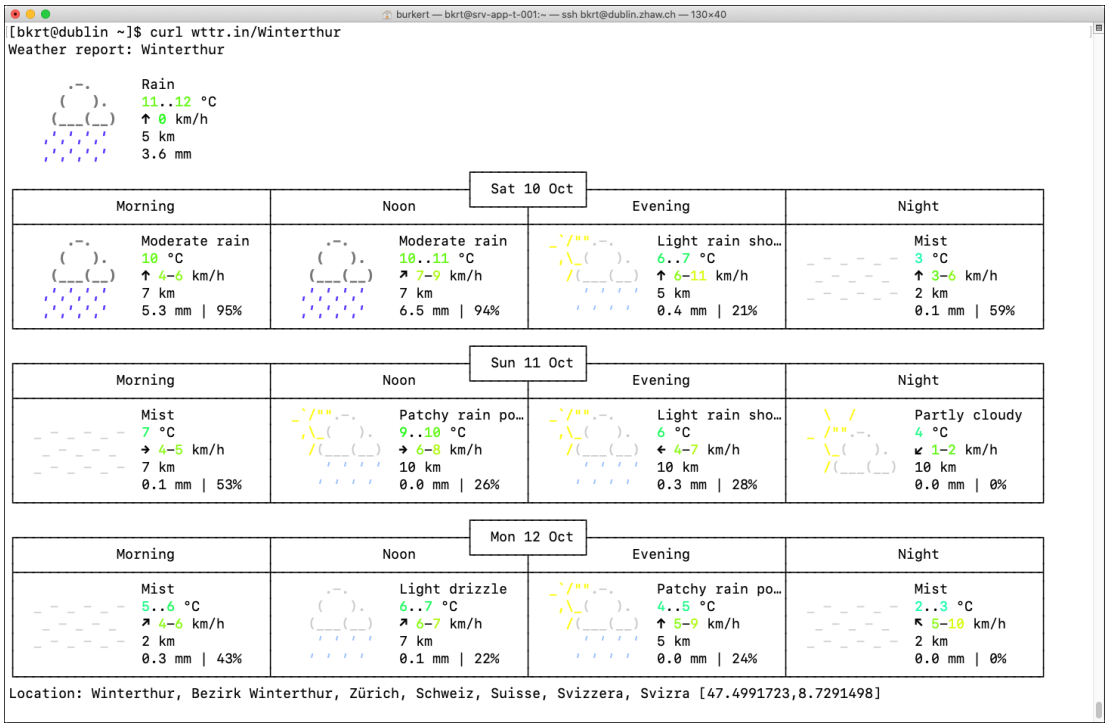
[1] Rekursiver Ansatz: Falls die Abbruchbedingung noch nicht erfüllt ist, wird der Startwert mit *cons* in die Liste eingefügt, die der rekursive Aufruf erzeugt. Was ist der neue Startwert im rekursiven Aufruf?

3. Quicklisp, HTTP-Zugriff und JSON

(fakultative Vertiefung)

Probieren Sie auf der Kommandozeile einmal folgenden Aufruf (falls *curl* nicht installiert ist, tut's auch der Browser, da wir nur GET-Requests benötigen):

```
$ curl wttr.in/Winterthur
```



Alternativ können die Wetterdaten auch in JSON abgefragt werden:

```
$ curl "wttr.in/Winterthur?format=j1"
```

Die soll nun verwendet werden, um eine Lisp-Funktion zu schreiben, welche die aktuelle Temperatur zu einem bestimmten Ort liefert. In den Standard-Funktionen von Common Lisp fehlen uns dazu die folgenden Funktionen:

- HTTP-Zugriff auf einen Server
- Parsen der JSON-Antwort

Beides lässt sich mit Hilfe eines Paketmanagers leicht nachrüsten. Am besten verwenden wir den Paketmanager *Quicklisp*. Das ist eine Paketverwaltung für die gängigen Common-Lisp-Installationen. Es liegt bisher erst in einer Betaversion vor.

Hinweis: Ich habe *Quicklisp* bisher mit *clisp* und *sbcl* getestet. Im Gegensatz zu früheren *Quicklisp*-Versionen hatte ich (Stand Oktober 2020) in *clisp* Probleme mit der Installation der Pakete, aber die aufgetretenen Fehler nicht weiter untersucht. Mit *sbcl* hat es problemlos geklappt.

Zur Installation wird die Datei *quicklisp.lisp* von der Website *http://beta.quicklisp.org* geladen und ausgeführt. Ablauf am Beispiel SBCL:

```
$ curl -O http://beta.quicklisp.org/quicklisp.lisp
$ curl -O http://beta.quicklisp.org/quicklisp.lisp.asc
```

An dieser Stelle wäre die Signatur der geladenen Datei zu überprüfen. Dann geht es weiter:

```
$ sbcl --load quicklisp.lisp
... diverse Ausgaben ...
* (quicklisp-quickstart:install)
... diverse Ausgaben ...
* (ql:add-to-init-file)
... damit Quicklisp immer zur Verfügung steht ...
```

Anstelle des letzten Schritts kann auch jeweils *quicklisp/setup.lisp* (relativ zum Homeverzeichnis) geladen werden. Damit ist Quicklisp installiert.

Nun werden alle Pakete mit „json“ im Namen aufgelistet und anschliessend werden die Pakete *cl-json* und *drakma* (für die HTTP-Requests) geladen:

```
* (ql:system-apropos "json")
#<SYSTEM cl-json / cl-json-20141217-git / quicklisp 2015-01-13>
#<SYSTEM cl-json.test / cl-json-20141217-git / quicklisp 2015-01-13>
...
* (ql:quickload "cl-json")
... ; Loading "cl-json" ...
* (ql:quickload "drakma")
... ; Loading "drakma" ...
```

Jetzt sind wir soweit, dass wir einen HTTP-Request machen können:

```
* (drakma:http-request "http://wttr.in/Winterthur?format=j1")
#(123 10 32 32 32 32 34 99 117 114 114 101 110 116 95 99 111 110 100 105 116
  105 111 110 34 58 32 91 10 32 32 32 32 32 32 32 32 123 10 32 32 32 32 32 ...)
```

Ein kleines Problem ist noch offen: Da *application/json* nicht als Textformat interpretiert wird, wird ein Bytevektor zurückgegeben. Wir ergänzen also den Content-Type:

```
* (setq drakma:*text-content-types* '(("text") ("application"."json")))
(("text") ("application" . "json"))
* (drakma:http-request "http://wttr.in/Winterthur?format=j1")
"{ \"current_condition\": [{ \"FeelsLikeC\": \"11\", ...}"
```

Schliesslich ist noch der JSON-String zu parsen. Dazu dient die Funktion *json:decode-json* des Pakets *cl-json*. Diese Funktion erwartet einen Stream als Input. Also machen wir aus dem String einen Stream:

```
* (let ((json-data (drakma:http-request "http://wttr.in/Winterthur?format=j1")))
    (with-input-from-string (s json-data)
      (json:decode-json s)))
((:CURRENT--CONDITION
  ((:*FEELS-LIKE-C . "10") (:*FEELS-LIKE-F . "50") (:CLOUDCOVER . "0") ...)))
```

Das Ergebnis ist eine Assoziationsliste, auf die wir bequem mit der *assoc*-Funktion zugreifen können, zum Beispiel zunächst mit dem Schlüssel :CURRENT--CONDITION.

Aufgabe (endlich…): Schreiben Sie ein kleines Lisp-Programm, das einen Ort abfragt und dann die aktuelle Temperatur an diesem Ort ausgibt:

```
$ sbcl --noinform --load current-temp.lisp --quit
...
Bitte Ort eingeben:
Kirkenes
Aktuelle Temperatur: -1 Grad
$
```

Eine kleine Funktion vereinfacht die Eingabe des Orts:

```
(defun prompt (msg)
  (format t msg)
  (terpri)
  (read-line))
```

Links:

https://www.quicklisp.org/beta/
https://edicl.github.io/drakma/
https://common-lisp.net/project/cl-json/cl-json.html

Eine Zusammenfassung der wichtigsten Entwicklungen rund um Common Lisp Stand 2020:
https://lisp-journey.gitlab.io/blog/state-of-the-common-lisp-ecosystem-2020/

Ähnliche Zusammenfassung, ein paar Jahre früher:
http://borretti.me/article/common-lisp-sotu-2015