Intelligente Suchverfahren

mit Python

KI mit Python

- In diesem Kurs werden auch Bereiche von *Algorithmen und Datenstrukturen* behandelt.
- Die Konzeption für den Kurs des erweiterten Anforderungsbereichs setzt auf die Anwendung des *funktionalen Paradigmas*.
- Dazu passend die Wahl einer funktionalen Programmiersprache Scheme (Racket) bzw Haskell.

KI mit Python

- Im grundlegenden Niveau sollte kein Wechsel der Programmiersprache erfolgen.
- Daher wird die mögliche Realisierung mit der Programmiersprache Python notwendig.
- Python ist keine funktionale Sprache, kann aber funktionale Elemente zur Verfügung stellen.
- Dabei entstehen aber einige kleine (?) Probleme.

Listen und Tupel

- Listen sind ein veränderlicher Sammlungstyp (*Sequenztyp*):

```
liste=[1,2,3,4,5]
liste[2]='drei'       # Indexzählung beginnt bei 0
liste -> [1,2,'drei',4,5]
```

- Tupel sind nicht veränderlich:

```
tupel=(1,2,3,4,5)
tupel[2]='drei'
TypeError: 'tuple' object does not support item
assignment
```

Rekursion

- Viele Probleme –gerade bei Suchverfahren– lassen sich rekursiv leichter beschreiben als iterativ.
- Ein einfaches Beispiel:

```
def entferne(element, liste):
    if liste==[]:
        return liste
    elif element==liste[0]:
        return liste[1:]
    else:
        return liste[:1]+entferne(element, liste[1:])
```

slices (*kein Problem!*)

- Das Erzeugen von Teillisten durch slices sind eine komfortable Möglichkeit:

```
def entferne(element, liste):
    if liste==[]:
        return liste
    elif element==liste[0]: # das eine erste Element
        return liste[1:]    # alle ab dem zweiten
    else:
        return liste[:1]+entferne(element, liste[1:])
            # „alle“ bis zum zweiten
```

Rekursion

- Die Standard-Wiederholungsstruktur bei Python ist die for – Schleife. Naheliegend ist daher eher eine iterative Programmierung.
- Rekursion ist möglich, aber (*wie üblich*) in der Rekursionstiefe begrenzt -> „stack overflow“
- Python erkennt keine Endrekursion und legt daher auch dann, wenn es nicht nötig wäre, die Aufrufe auf dem Stack ab.

call by reference

- Da Python (*nicht mit* call by value *wie bei Scheme*) bei Funktionsaufrufen nur eine Referenz (Zeiger) auf das übergebene Objekt an die Funktion weiter gibt, kann diese Funktion das Objekt auch für die aufrufende Stufe relevant verändern.

call by reference

- Will man das vermeiden, muss man von Objekten (*wie beispielsweise Listen*) eine echte Kopie erstellen.
  Beispiel: sortiere([]+Ausgangsliste) verändert die Ausgangsliste nicht.
- Gerade bei diesem Beispiel könnte die Veränderung sogar gewollt sein und wenn man das oft so gemacht hat, wird man das im anderen Fall leicht übersehen.

Übertragbarkeit

- Grundsätzlich gilt jedoch, dass funktionale Lösungen von Algorithmen bei Python sehr ähnlich den Scheme-Lösungen sind.
- Insbesondere sollte die Modularisierung und Beachtung der Kohärenz bei beiden Sprachen zu vergleichbaren Lösungen führen.