Stephan Jokiel
Maxine Rahlfes

# Programmieren 1 - WS 2020/21
# Übungsblatt 3

## Aufgabe 1: Skript

*a) Warum können Funktionen mit Seiteneffekt nicht wie Funktionen ohne Seiteneffekte durch algebraische Umformung evaluiert werden? Geben Sie ein Gegenbeispiel.*

Ein Seiteneffekt bedeutet,dass eine Funktion nicht nur mit den festgelegten Parametern arbeitet,  sondern mit Werten außerhalb der Funktion (Globale Daten oder aus einer Datei).

Durch das mehrfache Aufrufen einer Funktion mit Seiteneffekt ändert sich das Ergebnis, da Zwischenergebnisse außerhalb der Funktion gespeichert werden und mit ihnen dann wieder weitergearbeitet (weitergerechnet) wird (*siehe Datei "Bsp 1a"*).

*b) Welche Vorteile hat die cond-Anweisung gegenüber der if-Anweisung?*

Die Performance von cond ist ggf. besser als ganz viele verschachtelte if-loops. Saubererer Code, mehrere if/else-statements ineinander können schnell unübersichtlich werden und es ist nicht so einfach zu bearbeiten, wie conds.

*c) Welche Rolle spielen Symbole und Konstanten bei Intervallen?*

Mit Konstanten können die Intervallgrenzen übersichtlicher und verständlicher definiert werden. Außerdem können dadurch die Funktionen und Bedingungen vereinfacht werden.

Symbole geben Informationen darüber, wie die Daten der Funktionen in der "real word" zu interpretieren sind, wenn beispielsweise eine condition ausgeführt wird.

*d) Was war Ihnen beim Lesen der Kapitel 3-5 unklar? Wenn nichts unklar war, welcher Aspekt war für Sie am interessantesten?*

Etwas unklar war beim Lesen der Kapitel, was genau mit Seiteneffekten gemein ist und welche Probleme sie mit sich bringen.

**Aufgabe 2** wurde in der Datei "Aufgabe 2" bearbeitet.

**Aufgabe 3** wurde in der Datei "Aufgabe 3" bearbeitet.

Stephan Jokiel
Maxine Rahlfes

# Aufgabe 4: Schachposition

**Aufgabe 4 a- c** wurden in der Datei “Aufgabe 4” bearbeitet.

*d) Erläutern Sie kurz die Funktionsweise der Funktion knight-next-positions. Erläutern Sie insbesondere die letzte Zeile*

Knight_next_pos berechnet die nächstmögliche Position des Springers auf einem Schachbrett.
Dazu nimmt sie (die Funktion) eine Position “p” und wendet pos-to-point darauf an (macht also aus einem string (z.b. “B1”) ein Array (demzufolge [2 1]).
Mit den folgenden Zeilen (p .0 px! Und p .1 py!) speichert sie die Werte aus dem Array in px und py und speichert alle möglichen Züge (1 nach oben, 2 nach rechts, 2 nach oben, 1 nach rechts etc. pp) (durch “point-to-pos” in string-form gespeichert) in dem array “a”.

Das wohl bemerkenswerteste ist die letzte Zeile der Funktion: “[a { dup pos-valid not {pop} if } for] sort”.

Zunächst werden die gespeicherten Positionen (das Array a) aufgerufen und für (for-loop) jeden String in diesem Array die funktion pos-valid ausgeführt.
Für den Fall, dass (if-loop) diese Position nicht valide (pos-valid) ist, kommt ja zunächst erst einmal “false” auf den stack (da die position mit pos-valid angewandt “false” returned)
Durch das “not” wird dieses Ergebnis (true bei validem point, false bei invalid) umgewandelt, und mit dem {pop}-Operator der String aus “a” entfernt.
{pop} läuft nur, wenn “true” erscheint (if-loop).
Be einer validen Position gibt pos-valid ja bekannterweise “true” wieder.
Wenn jetzt true in der if-loop gelesen wird, würde {pop} den letzten Wert vom stack löschen (den zuletzt gelesenen string in “a” also).
Mit “not” wird “true” und “false” in “false” und “true” gewandelt. Der {pop}-Operator greift (if-loop) also nur bei Positionen, die von “pos-valid” als false ausgegeben wurden.