Unit 10b

Dr. Günter Kolousek

21. Juli 2015

Lege wiederum ein Verzeichnis an. Nennes es `15_unit10b`! In diesem Verzeichnis sollen alle Dateien der jeweiligen Einheit abgelegt werden.

1 Schulübungen

1. Schreibe ein Zahlenratespiel!

   Der Computer denkt sich eine Zahl zwischen 1 und 20 aus und es ist die Aufgabe des Benutzers diese zu erraten. Bei jedem Versuch wird mitgeteilt, ob die Zahl zu hoch oder zu nieder ist. Maximal 6 Versuche gibt es. Wird die Zahl nicht in 6 Versuchen erraten, dann wird eine entsprechende Meldung ausgegeben und das Programm beendet sich.

   Eine typische Benutzerinteraktion könnte so aussehen:

   ```
   Hallo! Wie lautet dein Name? Maxi
   Servus Maxi!

   Ich denke mir jetzt eine Zahl zwischen 1 und 20 aus.
   Wie lautet dein Tipp? 10
   Deine Zahl ist zu hoch!
   Wie lautet dein Tipp? 2
   Deine Zahl ist zu niedrig!
   Wie lautet dein Tipp? 4

   Gut geraten, Maxi! Du hast meine Zahl in 3 Versuchen erraten!
   ```

   Die ”zufällige” Zahl kann mittels der Funktion `randint` im Modul `random` ermittelt werden.

2. Schreibe eine Funktion `div_numbers(numbers, i1, i2)`, die ein Tupel `numbers` sowie zwei Zahlen `i1` und `i2` als Parameter bekommt und den Quotienten der zweier Zahlen berechnet und zurückliefert.

   Der Dividend wird durch den Index `i1` im Tupel ermittelt und der Divisor durch den Index `i2` im Tupel ermittelt.

   Tritt eine Division durch Null auf, dann soll ein NaN (not a number) Wert zurückgeliefert werden. Solch einen kann man in Python folgendermaßen erhalten:

   ```
   float("nan")
   ```

   Wird auf einen Index zugegriffen, der nicht exisitiert, dann soll `None` zurückgeliefert werden.

   Teste im interaktiven Interpreter!

3. Schreibe nun ein Programm `divide.py`, das den Benutzer nach zwei Indizes fragt und den Quotienten ausgibt.

   Als Inhalt des Tupels soll diese Folge von Werten verwendet werden::

   2 5 0 3 6 ”3” 1 9 8 0 2 8

   Tritt in der Funktion ein Fehler auf, dann soll dieser beim Aufruf mit einem `finally` abgefangen werden.

4. ”Rechne” mittels Python und der ”0b”-Notation in das Dezimalsystem um (analog dazu Oktal- und Hexadezimalsystem):

   a) 1111_2, 11001100_2, 111111_2

   b) 67_8, 123_8, 777_8

   c) FF_{16}, $FFFF_{16}$, $FFFFFFFF_{16}$

   Mache dich jetzt mit `int(n, b)` vertraut: Der erste Parameter ist die Zahl als String und der zweite Parameter ist die Basis als ganze Zahl! Überprüfe damit einige der vorhergehenden Ergebnisse.

5. ”Rechne” mittels Python und den eingebauten Funktionen `bin`, `oct` und `hex` die Zahlen 123, 456 und 789 (jeweils vom Dezimalsystem) in das entsprechende System um.

6. Rechne im interaktiven Interpreter mittels `&`, `|`, `^` und `~`:

   a) $1111_2 \wedge 0000_2$, $1100_2 \wedge 1100_2$

   b) $1111_2 \vee 0000_2$, $1010_2 \vee 0101_2$

   c) $1111_2 \veebar 0000_2$, $1010_2 \veebar 0101_2$

d) Die bitweise Negation ist abhängig von der Wortbreite und der Bedeutung in Programmiersprachen. Die Erklärung für das Verhalten folgt in einer späteren Einheit!

   Berechne:

   - ¬ 0, ¬ 1, ¬2, ¬3
   - ¬ -1, ¬ -2, ¬ -3, ¬ -4

7. Führe die folgenden Verschiebeoperationen aus:

   a) Verschiebe um jeweils 1 Bit nach links: 0, 1, 2, 3, 4, 5

   b) Verschiebe um jeweils 2 Bit nach links: 0, 1, 2, 3, 4, 5

   c) Verschiebe um jeweils 1 Bit nach rechts: 0, 1, 2, 3, 4, 5

   d) Verschiebe um jeweils 2 Bit nach rechts: 0, 1, 2, 3, 4, 5

8. Schreibe eine Funktion `bit(b, i)`, die das Bit am Index `i` des Byte `b` extrahiert und zurückliefert.

   Tipp: Erzeuge die Bitmaske durch Schieben eines Bits um `i` Bit nach links und verknüpfe die Bitmaske mit `b` mittels bitweisem AND.

9. Schreibe eine Funktion `set(b, i)`, die das Bit am Index `i` im Byte `b` setzt und zurückliefert.

   Tipp: Verwende dazu wiederum die Schiebeoperation und das bitweise OR.

10. Schreibe eine Funktion `clear(b, i)`, die das Bit am Index `i` im Byte `b` löscht und zurückliefert.

    Tipp: Verwende dazu wiederum die Schiebeoperation, das bitweise Invertieren und das bitweise AND.