**Repetition Python (Kapitel 7)** **Prüfungsstoff**

1. Du kannst mindestens zwei Vorteile von Funktionen für die Programmierung aufzählen.

   - Funktionen ermöglichen die *Wiederverwendung* von Programmcode.
   - Mit Funktionen lässt sich ein langes Programm in kleinere, besser lesbare Teile zerlegen (*Strukturierung*).
   - Sind Funktionen an einem zentralen Ort (einer *Funktionsbibliothek*) gespeichert, dann müssen Verbesserungen und Veränderungen dort nur einmal durchgeführt werden.

2. Du kannst die Syntax von Funktionsdefinitionen

   ```
   def funktionsname(parameterliste):
       funktionsblock
       return wert
   ```

   nachvollziehen und selber einfache Funktionen definieren.

3. Du kennst den Unterschied zwischen *Funktionsdefinition* und *Funktionsanwendung* und kannst die folgenden zwei Mechanismen (*Positionsparameter*, *Schlüsselwortparameter*) anwenden, mit denen die *formalen* durch die *aktuellen* Parameter ersetzt werden. *Beispiel:*

   ```
   def f(a, b, c):
       return 100*a + 10*b + c
   ```

   - *Positionsparameter:* `f(7, 1, 2)` ⇒ Wert: `712`
     Die aktuellen Parameter `(7, 1, 2)` werden in dieser Reihenfolge den formalen Parametern `(a, b, c)` zugewiesen.
   - *Schlüsselwortparameter:* `f(b=1, c=2, a=7)` ⇒ Wert: `712`
     Durch die Schlüsselwörter ist klar, welcher Wert zu welchem Parameter gehört. Daher können sie in irgend einer Reihenfolge stehen.

4. Du kannst erkennen, wenn in der Definition einer Funktion den formalen Parametern Standardwerte zugewiesen werden, so dass diese Werte verwendet werden, wenn beim Aufruf der Funktion der entsprechende Parameter fehlt.

5. Du weisst, dass die formalen Parameter und die innerhalb einer Funktion definierte Variablen nur innerhalb dieser Funktion (*lokal*) sichtbar sind und allfällige ausserhalb der Funktion definierte Variablen gleichen Namens während der Ausführung der Funktion „überschatten“. *Beispiel:*

   ```
   a = 3              # globales 'a'
   def f(x):
       a = 4          # lokales 'a'
       return x + a
   print(f(1))        # => 5 (innerhalb von f wird das lokale 'a' verwendet)
   print(a)           # => 3 (ausserhalb von f wird das globale 'a' verwendet)
   ```

   *Ausnahme:* Steht in der Parameterliste eine Variable, die auf ein veränderliches Objekt (z. B. eine Liste) verweist, dann wird zwar eine lokale Variable erzeugt, die jedoch auf das gleiche Objekt verweist, so dass jede das Objekt verändernde Operation, die auf eine der beiden Variablen angewendet wird, auch von der anderen Variablen „gesehen“ wird, da beide auf ein gemeinsames Objekt verweisen. Ist dieses Verhalten unerwünscht, muss man der Funktion eine echte Kopie des Objekts übergeben (z. B. eine Slice `L[:]` der ursprünglichen Liste `L`).

6. Du kannst Funktionen mit Rückgabewerten innerhalb zusammengesetzter Ausdrücken auswerten und du weisst, dass Funktionen ohne eine `return`-Anweisung den „leeren“ Wert `None` zurückgeben.

7. Du kannst einfache rekursive (sich selbst aufrufende) Funktionen nachvollziehen.