Praktikumsaufgabe 5 – Teil 1

Hinweis:

Im mitgelieferten Projekt finden Sie zu Teil 5.1 und 5.2 in den Tests die erwarteten Ergebnisse.

5.0 Themen

1. Funktionale und Objektrekursion
2. Einfach- und Endrekursion / Umformung in Rekursion mit Speicher

5.1 Funktionale Rekursion

1. Gegeben die iterative Näherungsformel für 1: $g(n) = \sum_{i=1}^{n} \frac{1}{i*(i+1)}$

   Formen Sie die Formel in eine rekursive Definition um und implementieren Sie die Formel rekursiv und endrekursiv.

2. Gegeben die iterative Näherungsformel für ½*ln(x) für x > 0:

   $\text{ln\_halbe}(x,n) = \sum_{i=0}^{n} \frac{(x-1)^{(2*i+1)}}{(2*i+1)*(x+1)^{(2*i+1)}}$, für $x > 0$

   Formen Sie die Formel in eine rekursive Definition um und implementieren Sie die Formel iterativ, rekursiv und endrekursiv.

3. Schreiben Sie eine rekursive Funktion, die prüft, ob eine Zeichenkette ein Palindrom ist. Die Methode soll Wort und Satz-Palindrome erkennen. Bei Satzpalindromen sollen die Satzzeichen, Anführungszeichen und Leer- und Sonderzeichen zuvor entfernt werden und alle Buchstaben klein dargestellt werden. Implementieren Sie die Methode auch endrekursiv.

5.2 Objektrekursion / strukturelle Rekursion

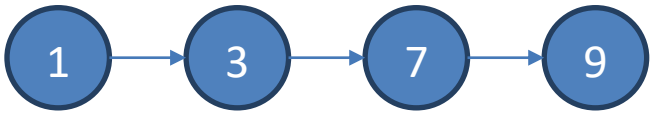
1. Gegeben ein beliebig geschachteltes Array. Berechnen Sie rekursiv das längste enthaltene Array. Methode `deep_max_width(ary)`.
2. Gegeben ein beliebig geschachteltes Array. Schreiben Sie die Methode `deep_count_elems_with_ary(ary)`, die für jedes enthaltene Array rekursiv die Anzahl der Elemente zählt und als Ergebnis ein Array von 2-elementigen Arrays zurückgibt, dessen 1'tes Element die Anzahl der Elemente und dessen 2'tes Element das Array ist.
3. Gegeben ein beliebig geschachteltes Array mit Elementen beliebigen Typs. Schreiben Sie die Methode `deep_group_by_type(ary)`, die die Elemente des

Arrays rekursiv nach Typ gruppiert. Das Ergebnis ist ein Hash, dessen Schlüssel die Typen und dessen Werte die Objekte des Typs sind.

4. Gegeben ein beliebig geschachtelter Hash, dessen Schlüssel und Werte wiederum Hashes sein können. Schreiben Sie die Methode `deep_reverse(a_hash)`, die die Schlüssel und Werte des Hashes rekursiv vertauscht.

5.3 Objektrekursion / einfach verkettete Liste

Eine einfach verkettete Liste besteht aus Knoten, die jeweils auf einen Nachfolgerknoten zeigen. Die Inhalte der Knoten sind die Elemente der Liste. In Abb. 1 sind die Inhalte der Liste 1,2,3,4.



1 Eine einfach verkettete Liste mit 4 Knoten. Die Elemente der Liste sind die Inhalte der Knoten

Eine Rahmen-Implementierung der Klassen `VerketteteListe` und `Knoten` ist in dem mitgelieferten Projekt bereits enthalten. Sie sollen die Klassen um die folgenden Methoden erweitern:

1. Implementieren Sie die Methode `empty?()` für die verkettete Liste.
2. Implementieren Sie die Methode `<<(inhalt)`, die `inhalt` an die Liste wie folgt anhängt:
   a. Ist die Liste leer, dann wird ein neuer Knoten mit `inhalt` erzeugt und `@start` zugewiesen.
   b. Sonst wird `<<` auf dem `@start` Knoten aufgerufen.

   Die Methode gibt als Ergebnis eine Referenz auf `self` zurück.
3. Implementieren Sie die Methode `<<(inhalt)` in der Klasse `Knoten`:
   a. Die Methode läuft an das Ende der Kette der Knoten. Das ist der Fall, wenn der Nachfolger eines Knotens `nil` ist.
   b. Und hängt am Ende einen neuen Knoten mit `inhalt` an.
4. Implementieren Sie für die Klasse `VerketteteListe` den Basis-Iterator, indem Sie diesen auf den Aufruf des Basis-Iterators der Klasse `Knoten` zurückführen.
5. Implementieren Sie für die Klasse `Knoten` den Basis-Iterator, der wie folgt arbeitet:
   a. wendet den Block auf den Inhalt des Knotens an.
   b. ruft den Iterator auf dem Nachfolger auf solange das Ende nicht erreicht ist.
6. Erweitern Sie die Klasse `VerketteteListe` durch Inkludieren eines geeigneten Moduls so, dass die Methode `to_s` der Klasse keinen Fehler mehr erzeugt.