1) Iteratoren **richtig falsch**

1.1) Iteratoren eignen sich hervorragend zum Bearbeiten aller Objekte einer Liste. Bei Sets können sie nicht verwendet werden.
1.2) Bei einem aktiven Iterator ist die Ablaufstruktur (z.B. WHILE-Schleife) bei jeder Anwendung des Iterators wieder zu implementieren, bei einem passiven Iterator nicht.
1.3) Man kann gleichzeitig mehrere Iteratoren verwenden.
1.4) Wenn man passive Iteratoren verwendet, braucht man oft auch globale Variablen. Dies kann bei rekursiven Aufrufen zu unbeabsichtigten Seiteneffekten führen.

falsch, richtig, richtig, richtig

2) Begriffe

2.1) Erklären Sie die Begriffe Sichtbarkeit und Lebensdauer von Variablen. Geben Sie ein Beispiel an, wo die Sichtbarkeit und die Lebensdauer einer Variable nicht gleich sind.

Sichtbarkeit: Bereich, in dem man über den Namen der Variable auf ihren Speicherplatz zugreifen kann.
Lebensdauer: Zeit, für die Speicherplatz für die Variable existiert.
Wenn eine globale Variable durch eine lokale Variable gleichen Namens überdeckt wird, ist sie nicht mehr sichtbar aber noch am Leben.

2.2) Erklären Sie die Begriffe globale, lokale und "intermediate" Variablen. Erklären Sie dabei, wo die verschiedenen Arten von Variablen sichtbar sind und was man über ihre Lebensdauer sagen kann.

Globale Variablen werden modulglobal deklariert, sind im ganzen Modul sichtbar (außer wenn sie durch lokale Variablen überdeckt werden), werden beim Laden des Moduls alloziert und leben, solange das Modul geladen ist.
Lokale Variablen werden innerhalb einer Prozedur deklariert und sind auch nur innerhalb dieser Prozedur (und innerhalb von lokalen Prozeduren dieser Prozedur) sichtbar, werden beim Aktivieren der Prozedur automatisch angelegt und beim Verlassen der Prozedur wieder freigegeben. Für jede Aktivierung einer Prozedur gibt es einen neuen Satz lokaler Variablen.
"Intermediate" Variablen sind lokale Variablen einer einschließenden Prozedur, wenn man sie von einer loakeln Prozedur aus betrachtet.

2.3) Erklären Sie die Begriffe statische, automatische und dynamische Daten.

Statische Daten sind modulglobale Variablen.
Automatische Daten sind lokale Variablen.
Dynamische Daten sind Daten, die mit NEW angelegt wurden. Wenn sie nicht mehr benötigt werden, wird ihr Speicherplatz vom Garbage Collector eingesammelt.

2.4) Erklären Sie den Unterschied zwischen einer konkreten und einer abstrakten Datenstruktur.

Eine konkrete Datenstruktur verbirgt keine Details. Klienten greifen direkt auf die Daten zu.
Der Klient kann leicht auf Kleinigkeiten bei den einzelnen Operationen vergessen.
Bei der Änderung der Datenstruktur müssen alle Klienten angepasst werden.
Klienten können Daten irrtümlich zerstören.

Eine abstrakte Datenstruktur ist ein Baustein, von dem nur die Operationen, nicht aber die Daten bekannt sind. Die Implementierung ist versteckt, von Details wird abstrahiert.
Abstrakte Datenstrukturen sind als Modul implementierbar.
Eine ADS ist für Klienten einfach zu benutzen. Klienten merken nichts von Änderungen der Datenstruktur. Klienten können die Daten nicht unbeabsichtigt zerstören.
Nachteil: Man sieht nicht genau, wie die Operationen implementiert sind.

3) ADS und ADT **richtig falsch**

3.1) Eine ADS kann in einem Modul implementiert werden, ein ADT nicht.
3.2) Wenn ein ADT in einem Modul implementiert wird, soll das Modul so heißen wie der ADT.
3.3) Ein Modul, das eine ADS implementiert, hat meist keine globalen Variablen.
3.4) Ein Modul, das einen ADT implementiert, hat meist keine globalen Variablen.

falsch, falsch, falsch, richtig

4) Dynamische Datenstrukturen

Implementieren Sie den ADT *Queue* für Zahlen mit den Operationen *Init, Insert, Remove*.

```
MODULE Queues;

TYPE
   Node = POINTER TO NodeDesc;
   NodeDesc = RECORD
      val: INTEGER;
      next: Node
   END ;

   Queue* = POINTER TO QueueDesc;
   QueueDesc* = RECORD
      head, tail: Node
   END ;

PROCEDURE Init* (q: Queue);
BEGIN
   NEW(q.head); q.tail := q.head
END Init;

PROCEDURE Insert* (q: Queue; val: INTEGER);
   VAR p: Node;
BEGIN
   NEW(p); p.val := val;
   q.tail.next := p; q.tail := p
END Insert;

PROCEDURE Remove* (q: Queue; VAR val: INTEGER): BOOLEAN;
BEGIN
   IF q.head = q.tail THEN RETURN FALSE
   ELSE q.head := q.head.next; val := q.head.val; RETURN TRUE
   END
END Remove;

END Queues.

Compiler.Compile @\s
```