3.2 Verkettete Listen

LK

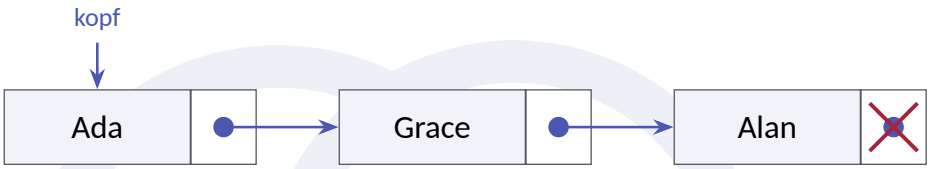
> LK
>
> Lineare Listen stehen nur im Lehrplan des Leistungskurses. Für den Grundkurs sind sie nicht verbindlich — als Vorübung zu den Bäumen aber nützlich, denn das Prinzip „Knoten mit Verweis" ist dasselbe.

Ein Feld hat eine feste Größe. Wer beim Programmieren `array [1..100]` schreibt, hat sich auf hundert Einträge festgelegt — der 101. passt nicht mehr hinein, und bei drei Einträgen liegen 97 Plätze brach.

Definition: Verkettete Liste

Eine **verkettete Liste** besteht aus **Knoten**. Jeder Knoten enthält einen Inhalt und einen **Verweis** auf den nächsten Knoten. Der letzte Knoten verweist auf *nichts*.

Ein Verweis auf den ersten Knoten heißt **Kopf** der Liste. Eine leere Liste ist eine, deren Kopf auf nichts verweist.



Jeder Knoten trägt einen Inhalt und einen Verweis auf den nächsten. Der letzte verweist auf nichts (`nil` bzw. `None`).

Eine verkettete Liste aus drei Knoten.

Die Knoten können im Speicher beliebig weit auseinanderliegen; zusammengehalten werden sie allein durch die Verweise. Dadurch wächst und schrumpft die Liste zur Laufzeit, ganz nach Bedarf.

Object Pascal — hier wird mit Zeigern gearbeitet:

```
type
  TListenZeiger = ^TListenKnoten;
  TListenKnoten = record
    inhalt: String;
    next: TListenZeiger;
  end;
```

Das Zeichen ^ steht für „zeigt auf". Bemerkenswert: `TListenZeiger` benutzt `TListenKnoten`, bevor dieser deklariert ist — bei Zeigertypen ist das ausnahmsweise erlaubt.

Python — dort ist jede Objektvariable ohnehin ein Verweis:

```
class ListenKnoten:
    def __init__(self, inhalt):
        self.inhalt = inhalt
        self.next = None
```

Stolperstein

Ab Abschnitt 3.3 kommt ein **zweiter** Knotentyp dazu: der Baumknoten mit *zwei* Verweisen. Beide heißen umgangssprachlich „Knoten", sind aber verschiedene Typen. Im Skript tragen sie deshalb verschiedene Namen.

3.2.1 Vorn einfügen

Am einfachsten wächst eine Liste am Kopf:

```python
def vorn_einfuegen(kopf, inhalt):
    neu = ListenKnoten(inhalt)
    neu.next = kopf          # der alte Kopf hängt sich hinten an
    return neu               # der neue Knoten ist jetzt Kopf
```

```
procedure VornEinfuegen(var kopf: TListenZeiger; inhalt: String);
var
  neu: TListenZeiger;
begin
  New(neu);
  neu^.inhalt := inhalt;
  neu^.next := kopf;
  kopf := neu;
end;
```

Gymnasium Ottweiler

Stolperstein

In der Pascal-Fassung ist die Reihenfolge der beiden letzten Zeilen entscheidend: Setzen Sie zuerst `kopf := neu`, ist der alte Kopf verloren — und mit ihm die ganze restliche Liste. **Erst anhängen, dann umhängen.**

Die Python-Fassung umgeht das, indem sie den neuen Kopf *zurückgibt*, statt ihn zu ändern. Der Aufrufer schreibt `kopf = vorn_einfuegen(kopf, "Ada")` — vergisst er die Zuweisung, passiert nichts.

3.2.2 Durchlaufen

```python
def ausgeben(kopf):
    aktuell = kopf
    while aktuell is not None:
        print(aktuell.inhalt)
        aktuell = aktuell.next
```

```
procedure Ausgeben(kopf: TListenZeiger);
var
  aktuell: TListenZeiger;
begin
  aktuell := kopf;
  while aktuell <> nil do
  begin
    WriteLn(aktuell^.inhalt);
    aktuell := aktuell^.next;
  end;
end;
```

Der Preis der Beweglichkeit zeigt sich hier: Um an den fünften Eintrag zu kommen, muss man vier Knoten abschreiten. Ein Feld liefert `feld[5]` sofort. **Die Liste kann wachsen, das Feld kann springen** — mehr ist der Unterschied nicht, und er entscheidet, welches passt.

Arbeitsauftrag 3

7. Schreiben Sie eine Anfrage `laenge(kopf)`, die die Zahl der Knoten liefert — einmal mit Schleife, einmal rekursiv.
8. Schreiben Sie `hinten_anhaengen(kopf, inhalt)`. Welchen Sonderfall müssen Sie zuerst behandeln?
9. Warum ist das Einfügen *vorn* billiger als das Einfügen *hinten*? Antworten Sie, ohne zu programmieren.