

1. (2 Punkte) Liste ist unser selbstgebastelter Datentyp. Wir nummerieren die Einträge der Liste mit Zeiger **anf** an der Stelle 0.
- Welche Zahlenfolge findet sich in den Listenlementen?
  - Gib den Index des Elements an, auf das der Zeiger **pos** zeigt.

```
a = Liste()
a.insert(1)
a.insert(2)
a.advance()
a.insert(3)
a.insert(4)
a.reset()
a.advance()
a.insert(6)
```

**Lösung:** a. 2 6 4 3 1      b. 1

2. (2 Punkte) Liste ist unser selbstgebastelter Datentyp. Wir nummerieren die Einträge der Liste mit Zeiger **anf** an der Stelle 0.
- Welche Zahlenfolge findet sich in den Listenlementen?
  - Gib den Index des Elements an, auf das der Zeiger **pos** zeigt.

```
a = Liste()
a.insert(1)
a.insert(2)
a.insert(3)
a.insert(4)
```

**Lösung:** a. 4 3 2 1      b. 0

3. (3 Punkte) Welche Ausgabe erscheint auf der Konsole?

```
class VerweisBox:
    def __init__(self, inhalt, unten=None, oben=None):
        inhalt: ein Zeichen
        unten, oben: eine Verweisbox
        '''
        self.inhalt = inhalt
        self.unten = unten
        self.oben = oben
        if unten is not None: self.unten.oben = self
        if oben is not None: self.oben.unten = self

    def __str__(self):
        return self.inhalt

a = VerweisBox('a')
b = VerweisBox('b')
c = VerweisBox('c', a, b)
d = VerweisBox('d', b, c)
print(a.unten)
print(b.unten.oben)
print(b.oben.oben.oben)
print(c.unten)
print(c.oben.oben.unten.unten.unten)
```

**Lösung:** None b b d d

4. (3 Punkte) Welche Ausgabe erscheint auf der Konsole?

```
class VerweisBox:
    def __init__(self, inhalt, unten=None, oben=None):
        self.inhalt = inhalt
        self.unten = unten
        self.oben = oben
        if unten is not None: self.unten.oben = self
        if oben is not None: self.oben.unten = self

    def __str__(self):
        return self.inhalt

a = VerweisBox('a')
b = VerweisBox('b')
c = VerweisBox('c', a, b)
e = VerweisBox('e', c, c)
print(a.oben)
print(b.oben)
print(b.unten)
print(c.oben.unten)
print(e.oben.unten.unten.oben)
```

**Lösung:** c None c c e

5. (4 Punkte) Die Klasse **FeatureListe** erbt von **Liste**. Sie hat zusätzlich ein Attribut **gerade**, das genau dann **True** ist, wenn die Anzahl der gespeicherten Listenelemente gerade ist. Schreibe die Klasse **FeatureListe**.

**Lösung:**

```
class FeatureListe(Liste):
    def __init__(self):
        super().__init__()
        self.gerade = True
    def insert(self, x):
        super().insert(x)
        self.gerade = not self.gerade
    def delete(self):
        super().delete()
        self.gerade = not self.gerade
```

6. (4 Punkte) Die Klasse **FeatureKeller** erbt von **Keller** und hat ein zusätzliches Feature. Die Methode **stuelp** kehrt die Reihenfolge der Elemente im Keller um. Wenn der Keller leer ist, wird ein **RuntimeError** geworfen. Der **FeatureKeller** soll kein zusätzlichen Attribut erhalten, aber für seine Arbeit darf er sich  $O(n)$  Zeit nehmen und er darf bei Bedarf einen unserer selbstgebastelten Datentypen **Liste**, **Keller** oder **Schlange** als Hilfe benutzen. Er darf leider keine in Python eingebaute Datenstruktur (wie z.B. **list**) benutzen.

**Lösung:**

```
class FeatureKeller(Keller):
    def stuelp(self):
        if self.empty(): raise RuntimeError("Der Keller ist leer")
        tmp = Schlange()
        while not self.empty():
            tmp.enq(self.top())
            self.pop()
        while not tmp.empty():
            self.push(tmp.front())
            tmp.deq()
```

7. (6 Punkte) Die Klasse **FeatureKeller** erbt von **Keller** und hat ein zusätzliches Feature. Die Methode **gerade** entscheidet, ob im Keller eine gerade Anzahl von Elementen liegen. Der Keller soll nach Ablauf der Methode unverändert sein. Wenn der Keller leer ist, wird ein **RuntimeError** geworfen. Der **FeatureKeller** soll kein zusätzlichen Attribut erhalten, aber für seine Arbeit darf er sich  $O(n)$  Zeit nehmen und er darf bei Bedarf einen unserer selbstgebastelten Datentypen **Liste**, **Keller** oder **Schlange** als Hilfe benutzen. Er darf leider keine in Python eingebaute Datenstruktur (wie z.B. **list**) benutzen.

**Lösung:**

```
class FeatureKeller(Keller):
    def gerade(self):
        if self.empty(): raise RuntimeError("Der Keller ist leer")
        tmp = Keller()
        zaehl = 0
        while not self.empty():
            tmp.push(self.top())
            self.pop()
            zaehl+=1
        while not tmp.empty():
            self.push(tmp.top())
            tmp.pop()
        return zaehl % 2 == 0
```