Programmierkurs Python I – WS 10/11

Übung 10

---

1 Cycle als Generator (2 + 2 + 2 Punkte)

Schreibe einen Generator `Cycle`, der die Elemente eines Sammelobjekts in unendlicher Folge aufzählt (siehe Übungsblatt 9).

```
>>> list(head(10, Cycle([1,2,3])))
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1]
```

Bonus #1: Schreibe das Programm so, dass es auch mit Mengen funktioniert. Die Reihenfolge der Elemente ist beliebig.

Bonus #2: Schreibe das Programm so, dass es auch Iteratoren als Eingabe akzeptiert.

2 Iterator mit Lookahead (3 Punkte)

Implementiere einen Iterator mit Lookahead, der die beiden folgenden Methoden zusätzlich zu `__iter__` und `__next__` implementiert:

- `lookahead(self)` liefert das nächste Element, ohne es zu „konsumieren“, oder wirft `StopIteration`, wenn kein nächstes Element vorhanden ist. Ein nachfolgenden Aufruf von `next` soll dasselbe Element liefern.
- `hasNext(self)` liefert `True`, wenn der Iterator noch ein nächstes Element hat, sonst `False`.

Beispiel:

```
>>> it = LookaheadIterator([1,2,3])
>>> next(it)
1
>>> it.lookahead()
2
>>> it.lookahead()
```

```
2
>>> next(it)
2
>>> it.hasNext()
True
>>> next(it)
3
>>> it.hasNext()
False
>>> it.lookahead()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 14, in lookahead
StopIteration
```

3 Merge (Bonus, 2 + 3 Punkte)

Die folgende Funktion nimmt zwei (Iteratoren über) sortierte Listen als Eingabe, und liefert einen Iterator über die sortierte Konkatenation der beiden Eingabelisten:

```
>>> def merge(xs, ys):
...   return sorted(list(xs) + list(ys))
...
>>> list(merge([1,3,5], [2,4]))
[1,2,3,4,5]
>>> list(merge(iter([1,3,5]), iter([2,4])
[1,2,3,4,5]
```

1. Implementiere eine äquivalente Variante von `merge`, die weder `sorted` verwendet noch intern mit Listen (`list(xs)`) arbeitet.

2. Erweitere die Funktion (aus Teil 1) so, dass beliebig viele Eingabe-Listen bzw. Iteratoren als Eingabe akzeptiert werden.

4 Binäre Bäume (Bonus, 3 Punkte)

Ändere die Beispiel-Implementierung von Binären Bäumen so, dass kein Generator sondern ein Iterator verwendet wird.

5 map (2 + 2 Punkte)

Reimplementiere die eingebaute Funktion `map`. Der Einfachheit halber kann davon ausgegangen werden, dass `map` genau 2 Argumente (Funktion + Iterable) hat.

Bonus: erweitere die Funktionalität so, dass beliebig viele Iterables als Argument übergeben werden können.

Zur Erinnerung: mit `def f(*args): ...` kann man Funktionen definieren, die beliebig viele Argumente nehmen. Wenn `args` eine Liste (mit `n` Elementen) ist, kann man eine `n`-stellige Funktionen `f` mit `f(*args)` auf die Liste anwenden.

Abgabe bis Donnerstag, 27.01.2011, 14:00 Uhr