Programmierkurs Python I – WS 10/11

Übung 4

---

1 Terminkalender (4 + 1 Punkte)

Implementiere eine Klasse, die eine sehr einfache Terminkalender-Funktionalität liefert. Die Klasse soll zwei Methoden haben: `addEvent(self,year,month,day,what)` soll einen Termin zu dem Kalender hinzufügen. Parameter sollen Jahr, Monat und Tag (als Zahlen) und ein String für den Termin-Inhalt sein.

Die zweite Methode `getEvents(self,year,month,day)` soll aus dem Kalender alle Termine für das angegebene Datum (Format wie oben) heraussuchen und als sinnvoll formatierten String zurückgeben.

Wähle sinnvolle Datenstrukturen, um die Termine im Terminkalender zu verwalten. Die `__init__`-Methode des Kalenders soll die leeren Datenstrukturen initialisieren:

```
class Calendar:
   def __init__(self):
      self.schedule =  ... % die leere Termin-Datenstruktur
```

Ein möglicher Programm-Ablauf könnte so aussehen:

```
>>> cal = Calendar()
>>> cal.addEvent(2010,11,11,"Python I, VL")
>>> cal.addEvent(2010,11,11, "CoLi Kolloquium")
>>> cal.addEvent(2010,12,24, "Weihnachten")
>>> print(cal.getEvents(2010,11,11))
1. Python I, VL
2. CoLi Kolloquium
```

Hinweis: Zeilenumbrüche in Strings erhält man, indem man ein `"\n"` einfügt.

Bonusaufgabe: Implementiere zwei zusätzliche Methoden `getEventsForMonth` und `getEventsForYear`, die alle Termine für einen gegebenen Monat bzw. ein gegebenes Jahr als String zurückliefern.

2 Tic Tac Toe (6 Punkte)

Ziel der Übung ist es, Tic Tac Toe als Python-Spiel für zwei Spielparteien zu implementieren; die Interaktion soll auf der Kommandozeile gesteuert werden. Tic Tac Toe wird auf einem Spielfeld von 3x3 Feldern gespielt, mit zwei Spielern die unterschiedliche Symbole benutzen. Gezogen wird abwechselnd; gewonnen hat, wer zuerst eine Reihe, Spalte oder Diagonale ganz mit seinen eigenen Symbolen füllt.
(Siehe auch: `http://de.wikipedia.org/wiki/Tic_Tac_Toe`)

Wir geben eine Klasse `GameBoard` vor, die verschiedene Funktionalitäten eines Spielbrettes implementiert: Initialisiert werden kann die Klasse mit dem Konstruktor und den Seitenlängen des Brettes (für Tic Tac Toe sind Höhe und Breite 3). Für Eure Methoden braucht ihr hauptsächlich die Methoden zum Setzen eines Symbols auf ein Feld und zum Abfragen, welches Symbol auf welchem Feld steht. Felder werden mit ihren Koordinaten benannt, bei 0 beginnend. (`(1 1)` ist z.B. das Feld in der Mitte des Tic-Tac-Toe-Brettes) Wir geben Euch die Symbole für die Spieler im Template for. `self.white` ist das Symbol für den Spieler, der anfangen darf, das andere ist `self.black`

Außerdem geben wir Euch ein `TicTacToe`-Template vor, dessen (noch leere) Methoden ihr implementieren müsst:

- `__init__` initialisiert zusätzlich zu unseren Vorgaben u.a. das Spielbrett und alle Hilfsstrukturen, die ihr benutzt.
- `isLegalMove(self ,x,y,s)` soll `True` zurückgeben, wenn man ein Symbol s auf das Feld (`x,y,s`) setzen darf, und sonst `False` (wenn das Feld besetzt ist oder nicht existiert).
- `move(self ,x,y,s)` setzt ein Symbol `s` auf das Feld `(x,y)`
- `evaluateBoard(self)` evaluiert das aktuelle Spielbrett. Die Methode soll -1 zurückgeben, wenn das Spiel noch nicht zu Ende ist; 0 zurückgeben, wenn der erste Spieler gewinnt; 1 für unentschieden; 2 wenn der zweite Spieler gewinnt.

Die Methode `play()` geben wir vor; sie nimmt Spielzüge von abwechselnden Spielern entgegen und verwertet sie, und gibt das aktuelle Brett aus so wie den Gewinner, falls das Spiel zu Ende ist. Sie benutzt die oben angegebenen Methoden. Ziehen kann man durch Eingabe des nächsten Feldes, wobei wir hier die Koordinaten in menschenlesbare konvertieren. (Wenn der Benutzer (2 2) eingibt, ist das die Brett-Mitte, intern ist das mittlere Feld, wie erwähnt, (1 1). Darum braucht ihr Euch nur dann zu kümmern, wenn ihr das Spiel **spielt**!) Aussehen kann das so:

```
+ ---- + ---- + ---- +
+      +      +  o   +
+ ---- + ---- + ---- +
+      +  x   +  x   +
+ ---- + ---- + ---- +
+  o   +      +  o   +
+ ---- + ---- + ---- +
x's Zug?
1 2
+ ---- + ---- + ---- +
+      +      +  o   +
+ ---- + ---- + ---- +
+  x   +  x   +  x   +
+ ---- + ---- + ---- +
+  o   +      +  o   +
+ ---- + ---- + ---- +
x gewinnt!
```

3 Listen-Elemente zählen (2 + 1 Punkte)

Schreibe eine Funktion `countElements(liste)`, die eine Liste als Parameter nimmt und für jedes Element der Liste zählt, wie oft es in der Liste vorkommt. Rückgabewert soll ein Dictionary sein, das als Schlüssel die unterschiedlichen Listenelemente enthält und als Werte Integer, die angeben, wie oft der Schlüssel in der Liste vorkam.

```
>>> liste = ['a', 1, 'a', 2, 1, 'a', 3]
>>> print(countElements(liste))
{'a': 3, 2: 1, 1: 2, 3: 1}
```

Bonusaufgabe: Implementiere die Funktion so, dass sie kein Dictionary, sondern eine Liste von Tupeln zurück gibt. Die Tupel sollen Paare sein, die genauso aufgebaut sind wie Schlüssel und Werte des ursprünglichen Dictionaries (erstes Element im Tupel = Element aus der Liste, zweites Element = Anzahl des Vorkommens). Die zurückgegebene Liste soll sortiert sein, und zwar absteigend nach der Vorkommens-Anzahl der Listenelemente (also nach den zweiten Elementen der Tupel).

```
>>> print(countElements(liste))
[('a',3), (1, 2), (2, 1), (3,1)]
```

Abgabe bis Donnerstag, 18.11.10, 14:00 Uhr per Mail an
`regneri@coli.uni-sb.de` und
`stth@coli.uni-sb.de`