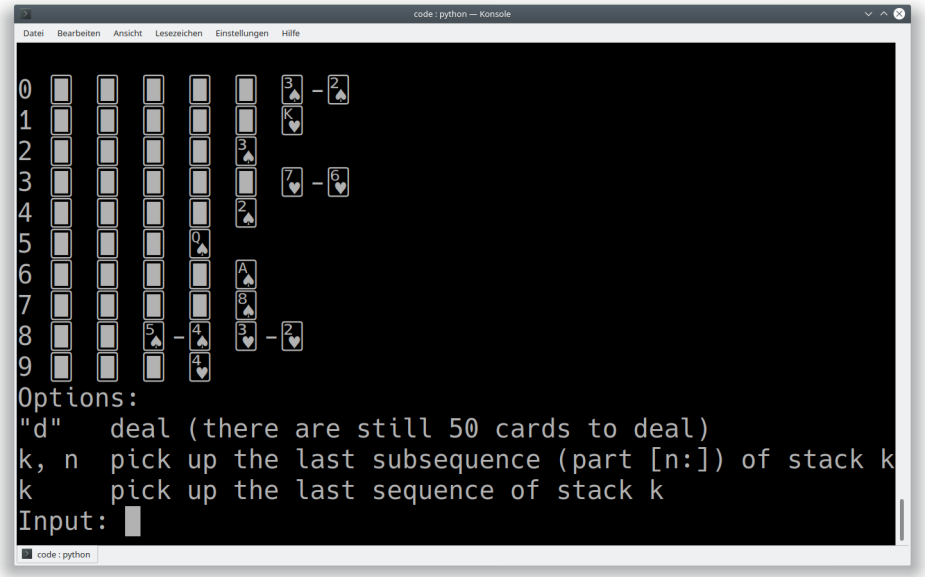
Institut für Angewandte und Numerische Mathematik
Sommersemester 2021
Dr. Daniel Weiß, M.Sc. Kevin Ganster

Einführung in Python

Spider Solitaire

Das Spiel *Spider Solitaire*

Ziel dieses Übungsblatts ist eine auf der Konsole spielbare objektorientierte Implementierung der Spiellogik von Spider Solitaire.



Neben der Klasse `Card`, welche wir bereits in Aufgabe 3 des 5. Übungsblatts implementiert haben, wollen wir die folgenden zusätzlichen Klassen einführen:

a) die Klasse `Sequence`, welche eine Sequenz beschreibt,

b) die Klasse `Stack`, welche einen Stack (Stapel von Sequenzen) beschreibt und

c) die Klasse `SpiderSolitaire`, die das Spider Solitaire Spiel als Ganzes verwaltet.

Die Klasse `Sequence`

Die Klasse `Sequence` besitzt nur das **Listen-Attribut** `_cards`.

Der **Konstruktor** erwartet eine Liste von `Card`-Instanzen, die im Attribut `_cards` gespeichert wird. Zusätzlich soll im Konstruktor geprüft werden, ob es sich bei der übergebenen Liste, wirklich um eine Sequenz handelt. Sie können dazu den Code aus der Funktion `is_sequence` von Aufgabe 3 des 4. Übungsblatt nutzen ohne jedoch eine eigene Methode zu schreiben. Falls es sich nicht um eine Sequenz handelt, soll der Konstruktor dies über einen `print`-Aufruf mitteilen und mit `return` unmittelbar beendet werden.
Erinnerung: Wir bezeichnen eine nichtleere absteigende Folge von passenden Karten einer Farbe als Sequenz.

Wir definieren eine Reihe weiterer **Methoden**, welche letztlich alle auf der Funktionalität des Datentyps `list` beruhen. Dies erzeugt einerseits etwas Programmieraufwand, resultiert aber in aussagekräftigen Namen der Methoden und in einem sauberen Kapselungsprinzip der Daten. So wird z. B. durch die Methode

```
def first_card(self):
    "Liefert die erste Karte dieser Sequenz"
    return self._cards[0]
```

letztlich nur der Zugriffsoperator für Listen genutzt. Der direkte Zugriff auf das Attribut `_cards` von außen ist dadurch jedoch nicht nötig. Durch diese Schnittstelle wird nach außen nicht deutlich, dass die interne Datenstruktur durch eine Liste realisiert ist.

Weitere **Methoden** dieser Klasse sind

- `last_card`, siehe `first_card`.
- `is_full`, welche einen booleschen Wert liefert, ob diese Sequenz bereits vollständig ist, d.h. aus 13 Karten besteht.
- `fits_to`, die im Aufbau und Funktionalität der `fits_to`-Methode der `card`-Instanzen auf Ebene von Sequenzen entspricht und diese für passenden `Card`-Instanzen aufruft (siehe Aufgabe 3 des 4. Übungsblatts).
- `merge`, welche eine andere Sequenz entgegennimmt und die Karten der anderen Instanz zur eigenen Sequenz hinzufügt. Hier soll mit Hilfe von `fits_to` überprüft werden, ob das Hinzufügen erlaubt ist. Ist dies nicht möglich soll wiederum durch einen `print`-Aufruf darauf hingewiesen werden und die Methode mit `return` unmittelbar beendet werden.

- `split`, welche einen Index entgegennimmt, den hinteren Teil der Sequenz an diesem abtrennt und den abgetrennten Teil als eine neue Sequenz zurückgibt. Ist der Index kleiner oder gleich 0 oder größer oder gleich `len(self._cards)`, so soll durch einen `print`-Aufruf mitgeteilt werden, dass das Abtrennen nicht durchgeführt wird, und die Methode mit `return` unmittelbar beendet werden.. *Beachte: Wäre der Index gleich 0 und würden wir die Sequenz an diesem Index abtrennen, bliebe eine leere Liste zurück. Dies ist aber ein inkonsistenter Zustand einer Sequenz.*
- `__str__`, welche eine Repräsentation der Sequenz als String zurückliefert.

Die Klasse `Stack`

Die Klasse `Stack` verwaltet folgende **Attribute**:

- `_sequences`, das die in diesem Stack enthaltenen Sequenzen als Liste der `Sequenz`-Instanzen speichert und verwaltet.
- `_face_down_cards`, das die Liste der noch verdeckten Karten des Stacks speichert.

Der **Konstruktor** erwartet eine `Card`-Instanz, welche die bereits sichtbare Karte repräsentiert und eine Liste von `Card`-Instanzen, welche die noch verdeckten Karten darstellen. Letztere wird im Attribut `_face_down_cards` gespeichert. Für die bereits sichtbare Karte wird eine neue Liste mit einer `Sequence`-Instanz erstellt und in `_sequences` gespeichert. Beachten Sie hierfür den Konstruktor von `Sequence`.

Des Weiteren werden folgende **Methoden** bereitgestellt:

- `is_empty`, die prüft, ob dieser Stapel leer ist, und in diesem Fall `True` zurückgibt. Ein Stapel ist leer, wenn keine Sequenz auf ihm liegt, hierbei spielen die verdeckten Karten keine Rolle.
- `last_sequence`, welche die letzte Sequenz innerhalb des Stapels zurückgibt. Prüfen Sie mit `is_empty`, ob es überhaupt eine solche Sequenz gibt. Falls nicht „`print` und `return`“ wie oben bereits mehrfach beschrieben.
- `append_sequence`, die eine Instanz von `Sequence` entgegennimmt und diese an `_sequences` anhängt.
- `remove_last_sequence`, welche die letzte Sequenz löscht. Prüfen Sie mit `is_empty`, ob es überhaupt eine solche Sequenz gibt. Falls nicht, wieder „`print` und `return`“.
- `test_revealcard`, die überprüft, ob es keine offenen aber noch verdeckte Karten auf diesem Stapel gibt, falls ja, wird eine Karte aufgedeckt und in einer `Sequence`-Instanz dem `_sequences`-Attribut durch `append_sequence` hinzugefügt.
- `test_fullsequence`, die überprüft, ob die letzte Sequenz in `_sequences` voll ist. Falls ja, wird diese Sequenz aus `_sequences` entfernt und mit `test_revealcard` überprüft, ob eine verdeckte Karte aufgedeckt werden muss.
- `deal_card`, die eine ausgeteilte Karte in Form einer `Card`-Instanz entgegennimmt und als einelementige Sequenz weiterverarbeitet. Falls diese Sequenz an die letzte Sequenz des Stapels passt, wird diese der letzten Sequenz hinzugefügt und mit `test_fullsequence` getestet, ob hierdurch die Sequenz voll ist. Falls die einelementige Sequenz nicht an die letzte Sequenz passt, wird sie als neue Sequenz dem Stapel hinzugefügt.
- `__str__`, die eine String-Repräsentation des Stapels zurückliefert.

Die Klasse `SpiderSolitaire`

Diese Klasse besitzt folgende **Attribute:**

- `_stack2deal`, eine Liste von `Card`-Instanzen, der noch zu verteilenden Karten.
- `_stacks`, eine Liste von zehn `Stack`-Instanzen.
- `moving_sequence`, speichert die Sequenz, die verschoben werden soll.
- `origin_stack_index`, in welchem der Index des Stapels gespeichert wird, von welchem eine (Teil-)Sequenz zum Verschieben entnommen wird. Dieser Index wird benötigt, falls das Verschieben nicht möglich ist und der Vorgang rückgängig gemacht werden muss.

Der Konstruktor besitzt neben `self` keinen weiteren Parameter, stellt die Ausgangssituation des Spiels her und lässt `moving_sequence` und `origin_stack_index` die Instanz `None` referenzieren. Um die Menge aller Karten zu mischen, wird die Funktion `shuffle` des Moduls `random` genutzt.

Weitere **Methoden** sind:

- `deal`, welche für jeden Stack eine Karte von `_stack2deal` austeilt. Hierfür soll die `deal_card` der `Stack`-Instanzen und `_stack2deal.pop()` benutzt werden. Dies ist jedoch nur möglich, wenn es noch Karten zu verteilen gibt (d.h. das Attribut `_stack2deal` nicht leer ist) und jeder Stapel mindestens eine aufgedeckte Karte beinhaltet (hierfür soll `is_empty` der einzelnen Stacks benutzt werden).
- `pick_up`, welche die Parameter `stack_index` und `card_index` aufweist, durch welche kommuniziert wird, von welchem Stack eine (Teil-)Sequenz zum Verschieben gewählt werden soll. Diese Methode ruft die `split`-Methode der Sequenz-Instanz auf, deren (Teil-)Sequenz verschoben werden soll, und setzt die Attribute `moving_sequence` und `origin_stack_index` entsprechend.
- `abort_move`, ohne zusätzlichen Parameter (außer `self`) wird hier das Zurück-

legen der aufgenommenen (Teil-)Sequenz basierend auf den Attributen `moving_sequence` und `origin_stack_index` im Fall eines nicht erfolgreichen oder nicht gewollten Verschiebens organisiert und diese Attribute wieder mit `None` belegt.

- `move`, welche einen Parameter `stack_index` besitzt, welcher den Index des Zielstapels kennzeichnet. Abhängig davon, ob der Zielstapel leer ist oder die zu verschiebende Sequenz neben dem Wert auch bezüglicher der Farbe an den Zielstapel passt oder eben nicht, wird die Sequenz an den Zielstapel angelegt. Im Fall des Zusammenfügens von Sequenzen muss anschließend überprüft werden, ob die Ergebnissequenz vollständig ist. Sollte die zu verschiebende Sequenz nicht an den Zielstapel angelegt werden können, so soll die Methode `abort_move` aufgerufen und dies zusätzlich durch einen `print`-Aufruf mitgeteilt werden.
- `is_won`, prüft, ob alle Stapel leer sind.
- `play`, organisiert die Interaktion mit dem User und somit das Spielen.
- `__str__`, welche eine Repräsentation des Spiels als String zurückliefert. Hierfür wird die Funktion `str` angewandt auf die `Stack`-Instanzen benutzt.

Aufgabe:

Implementieren Sie die oben genannten Konstruktoren, Attribute und Methoden in der Datei `spidersolitaire.py`. Gehen Sie dabei folgendermaßen vor:

a) Vervollständigen Sie zunächst die Klasse `Sequence`, indem Sie die in der Datei `spidersolitaire.py` fehlenden Methoden dieser Klasse implementieren. Gehen Sie nach der Reihenfolge vor, in der die Methoden oben beschrieben sind. Sie können mit Hilfe der Tests in der Datei `test_cases.py` Ihren Fortschritt überprüfen.

b) Gehen Sie analog für die Klasse `Stack` vor.

c) Implementieren Sie in der Klasse `SpiderSolitaire` die Methode `deal` und vervollständigen Sie die Methode `move` um das eigentliche Verschieben bzw. Abbrechen des Verschiebevorgangs.

Sie finden im ILIAS ein Video, welches das vorgegebene Template erklärt. Mit Hilfe der bereits genannten Datei `test_cases.py` können Sie überprüfen, ob ihre Implementierung den Anforderungen genügt. Hierzu finden Sie ebenfalls ein Video im ILIAS. Der Testcode deckt viele Teilimplementierungen ab. Deshalb ist es ratsam diesen möglichst früh und oft auszuführen.

Die Abgabe besteht aus der Datei `spidersolitaire.py`, welche wegen den Testfällen nicht umbenannt werden soll. **Eine bestandene Abgabe erzeugt bei Ausführung keinerlei Fehler und erfüllt alle Tests der** `test_cases.py`. Sollten während Ihrer Implementierung Fehler auftreten, die Sie nicht selbst beheben können, nutzen Sie bitte die Tutorien und stellen Sie Fragen.

Hinweis: Das Erhöhen der Konsolen-Schriftgröße kann helfen die Unicode-Zeichen für die Karten in der Ausgabe besser zu erkennen. Sie können auch mit der Tastenkombination „`Strg +`“ ranzoomen.

Material: ILIAS