Danny Rehl, Programmieren I - Python

Übungsblatt 04

Abgabefrist: Zusammen mit dem letzten Übungsblatt im Semester

---

Sie finden für dieses Übungsblatt eine Vorlage für Ihre Abgabe zum Herunterladen im Moodle.

Für die Bewertung Ihrer Abgabe zählen die von Ihnen vervollständigten Funktionen im oberen Teil der Vorlage. Schreiben Sie hier nur Ihre Lösungen hin, keine weiteren Befehle außerhalb der Funktionen.

Im unteren Teil finden Sie einen `if __name__ == "__main__"`-Abschnitt, welcher bereits einige Beispielaufrufe zu den Funktionen beinhaltet. Hier können Sie die Befehle beliebig ergänzen oder ersetzen, um Ihre Funktionen zu testen.

Hinweis: Trotz aller Tests muss die hochgeladene `.py`-Datei natürlich immer noch ausführbar bleiben. Beispielsweise wird ein `SyntaxError` im `if __name__ == "__main__"`-Teil in jedem Fall zur Pytest-Bewertung von 0 % führen.

Bonusübungsblatt

Sie werden im Laufe der untenstehenden Aufgaben Ihre eigene Implementation von Vier gewinnt schreiben. Auch wenn das vielleicht erstmal sehr aufwändig klingt, lassen Sie sich nicht entmutigen: Durch Zerlegung in Teilprobleme reduziert sich die vermeintliche Schwierigkeit des Projekts deutlich. Sie benötigen am Ende nur Code, den Sie auf den vergangenen oder kommenden Übungsblättern bereits ähnlich angewandt haben oder anwenden werden.

Die Abgabefrist dieses Übungsblatts stimmt mit der Abgabefrist des letzten Übungsblatts des Semesters überein. Sie können also das ganze Semester über immer wieder eine neue Aufgabe bearbeiten und die neuste Version auf Tutron hochladen.

Da die Aufgaben teilweise aufeinander aufbauen, ist es wichtig, dass Sie die Aufgaben in der angegebenen Reihenfolge bearbeiten.

In der Vorlage finden oberhalb der gewohnten Abschnitte dieses Mal zusätzlich folgenden umfangreichen Teil:

```
# Vorgegebener Code (Hier nichts verändern)
```

Sie sollen in diesem Abschnitt, wie es im Kommentar steht, nichts am Code ändern. Das soll aber nicht bedeuten, dass Sie den Abschnitt ignorieren sollen oder können. Es wird sich für das Verständnis zur Bearbeitung der Aufgaben lohnen, den Code genau durchzuschauen.

Aufgabe 1 (1 Bonuspunkt)

Wir benötigen zunächst eine Repräsentation der Spieldaten, sodass wir mit unserem Code daran arbeiten können. Das Spielbrett besteht aus einem Raster mit 6 Reihen (Zeilen) und 7 Spalten. Es kann daher sehr anschaulich von einer 6x7 Matrix dargestellt werden. Diese Matrix speichern wir im Code als mehrdimensionale Liste `grid`.

Vervollständigen Sie die Funktion `empty_grid`, welche als Rückgabewert eine mehrdimensionale Liste hat. Die Elemente der ersten Dimension sind die 6 Reihen des Spielbretts, jede Reihe (also die zweite Dimension) enthält wiederum 7 Elemente von Typ `int` mit Wert `0`.

Die Liste als Rückgabewert sollte also so aussehen:

```
[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]]
```

Achten Sie darauf, dass Sie Reihen- und Spaltenanzahl richtig herum wählen. Mit `grid[0][0]` soll man auf den Wert des Feldes ganz links oben zugreifen können, mit `grid[5][6]` auf den des Feldes ganz rechts unten. (Da der Index bei 0 beginnt also die 6. Reihe, 7. Spalte.)

Wenn Sie Reihen- und Spaltenanzahl aus Versehen vertauschen, führt `grid[5][6]` zu einem `IndexError`.

Aufgabe 2 (1 Bonuspunkt)

Vervollständigen Sie die Funktion `print_grid`. Sie soll die in `grid` gespeicherten Spieldaten ausgeben. Dabei soll der Wert `0` durch ein Leerzeichen repräsentiert werden, Spieler 1 mit Wert `1` durch ein `X` und Spieler 2 mit Wert `2` durch ein `O`. Um die Felder herum zeichnen Sie mit `|`, `+` und `-` einen Rand. Die einzelnen Spalten haben einen Abstand von 1 Leerzeichen. Unter der unteren Begrenzung des Spielfelds sind die Spalten mit den Zahlen `1` bis `7` beschriftet.

Schauen Sie sich die vorläufige Ausgabe der Funktion an und erweitern Sie diese Stück für Stück bis sie alle geforderten Ansprüchen erfüllt.

Hinweis: Sie benutzen an dieser Stelle den Parameter `end` der `print`-Funktion:

```
print(text, end="")
```

Das gibt die Variable `text` aus ohne anschließend automatisch einen Zeilenumbruch `\n` anzuhängen. Schauen Sie gegebenenfalls nochmal in die Folien der Vorlesung oder in die Python-Dokumentation zu `print` für nähere Informationen.

Um Ihre Funktion zu testen, können Sie die in der Vorlage enthaltenen Beispiele `EXAMPLE_GRID_1`, `EXAMPLE_GRID_2` und `EXAMPLE_GRID_3` ausgeben lassen. Sie sollten die folgenden Ausgaben erhalten:

```
|               |
|               |
|               |
|               |
|               |
|               |
+---------------+
  1 2 3 4 5 6 7
```

```
|     X         |
|     O         |
|     X         |
| O   O         |
| X   X X     O |
| X O O X   O X |
+---------------+
  1 2 3 4 5 6 7
```

```
| O O X X O O O |
| O X O O X X X |
| X O X O X O O |
| O X O X X X O |
| X O O X O X O |
| X X O O X X X |
+---------------+
  1 2 3 4 5 6 7
```

Aufgabe 3 (1 Bonuspunkt)

Die Werte der Matrix, welche das Spielbrett repräsentiert, sind in `grid` jeweils in Reihen abgelegt. Um eine Liste mit den Werten einer Spalte der Matrix zu erhalten ist eine Funktion nötig.

Vervollständigen Sie die Funktion `get_column`, sodass die mit `column_number` angegebene Spalte der Matrix zurückgegeben wird. (Werte von oben nach unten.)

Beispiel: Für `get_column(0, EXAMPLE_GRID_3)` erwarten wir den Rückgabewert `[2, 2, 1, 2, 1, 1]`. (Vergleiche auch Ausgabe von `EXAMPLE_GRID_3` oben.)

Aufgabe 4 (1 Bonuspunkt)

In eine Spalte passen 6 Werte. Wenn ein Spieler sich für seine "Scheibe" eine Spalte ausgesucht hat, muss überprüft werden, ob in dieser Spalte noch ein Feld frei ist und falls ja, an welcher Stelle. (Die Scheiben müssen schließlich von unten nach oben angeordnet sein.)

Vervollständigen Sie die Funktion `free_space`, sodass die mit `column_number` angegebene Spalte auf ein freies Feld überprüft wird. Ist kein Feld mehr frei, wird `None` zurückgegeben. Ist noch ein Feld frei, dann wird der Index zurückgegeben, mit dem man in `grid` auf diese Reihe zugreifen kann. (Also der Index derjenigen Reihe, welche in der angegebenen Spalte von unten gesehen als erstes ein freies Feld besitzt, beziehungsweise von oben gesehen das letzte freie Feld enthält.)

Die Funktion benutzt `get_column` aus Aufgabe 3, um die Werte der angegebenen Spalte zu erhalten. (Achtung, Hinweis: Die Funktion gibt Ihnen die Werte von oben nach unten zurück. Freie Felder müssen aber von unten nach oben gesucht werden.)

Beispiele: Für `free_space(1, EXAMPLE_GRID_2)` erwarten wir `4`. Für `free_space(2, EXAMPLE_GRID_2)` erwarten wir `None`. (Vergleiche auch Ausgabe von `EXAMPLE_GRID_2` oben.)

Aufgabe 5 (1 Bonuspunkt)

Um herauszufinden, ob das Spiel in einem Unentschieden geendet ist, prüfen wir, ob auf dem Spielbrett überhaupt noch irgendwo freie Felder sind.

Vervollständigen Sie die Funktion `grid_is_full`, sodass sie `True` zurückgibt, wenn kein Feld mehr frei ist - und entsprechend `False`, wenn noch mindestens ein Feld frei ist. Schreiben Sie dazu eine Schleife über alle Spalten: wenden Sie `free_space` aus Aufgabe 4 auf jede Spalte an. Sollte `free_space` für alle Spalten `False` sein, ist kein Feld mehr frei. Oder auch andersherum gesagt: sollte `free_space` für mindestens eine Spalte `True` sein, ist noch mindestens ein Feld frei.

Beispiele: `grid_is_full(EXAMPLE_GRID_1)` und `grid_is_full(EXAMPLE_GRID_2)` sind `False`. Hingegen erwarten wir `True` für `grid_is_full(EXAMPLE_GRID_3)`. (Vergleiche auch Ausgabe von `EXAMPLE_GRID_1`, `EXAMPLE_GRID_2` und `EXAMPLE_GRID_3` oben.)

Aufgabe 6 (1 Bonuspunkt)

Vervollständigen Sie die Funktion `drop_disc`, sodass eine "Scheibe" in die mit `column_number` angegebene Spalte eingeworfen wird. Der aktuelle Spieler (also der Wert, den Sie in `grid` eintragen müssen) ist als Parameter `player` übergeben.

Benutzen Sie `free_space` aus Aufgabe 4, um herauszufinden, in welcher Reihe Sie `grid` verändern müssen. Falls Sie beim Aufruf von `free_space` jedoch `None` zurückbekommen, bedeutet das, dass in der vom Spieler gewählten Spalte kein Feld mehr frei ist. Die Funktion `drop_disc` soll in diesem Fall `False` zurückgeben. Falls der Wert hingegen erfolgreich gespeichert werden konnte, geben Sie `True` zurück.

Beispiel: `drop_disc(2, 1, EXAMPLE_GRID_2)` ist `False`. Hingegen erwarten wir für `drop_disc(1, 1, EXAMPLE_GRID_2)` ein `True` und können uns das veränderte `EXAMPLE_GRID_2` hinterher mit `print_grid` anschauen.

Aufgabe 7 (1 Bonuspunkt)

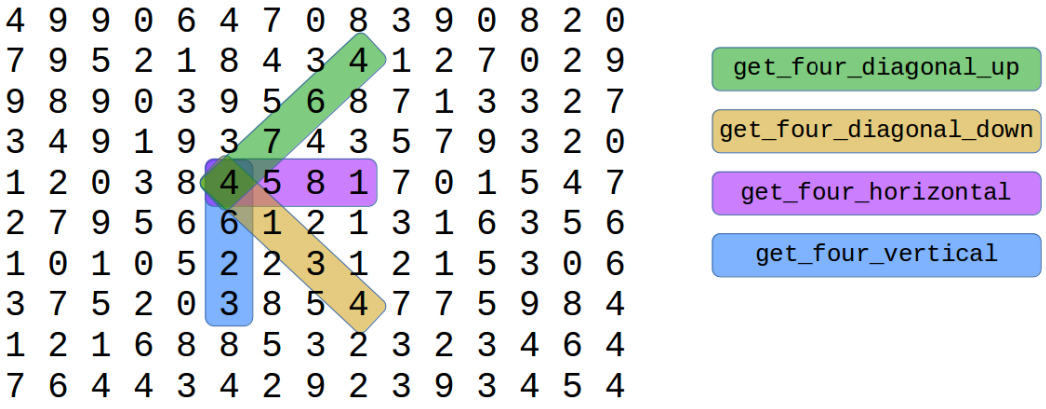
Vervollständigen Sie die Funktion `all_elements_equal`. Diese Funktion soll prüfen, ob alle Elemente der als Parameter `sequence` übergebenen Liste gleich sind. Falls ja, wird `True` zurückgegeben - falls nicht, entsprechend `False`. Für den Fall, dass Parameter `sequence` keine Liste ist, sondern den Wert `None` enthält, soll auch `False` zurückgegeben werden.

Beispiele: Für `all_elements_equal([1, 2, 3])` erwarten wir `False`. Der Sonderfall `all_elements_equal(None)` soll ebenfalls `False` sein. Hingegen sind `all_elements_equal([1, 1, 1])`, `all_elements_equal([True, True, True])` oder `all_elements_equal([False, False, False])` alle `True`.

Aufgabe 8 (1 Bonuspunkt)

Um zu überprüfen, ob ein Spieler gewonnen hat, benötigen wir von einem bestimmen Spielfeld ausgehend in vier Richtungen die Werte der vier angrenzenden Felder - und zwar diagonal hoch und runter, sowie horizontal und vertikal.

Das ausgehende Spielfeld wird dabei über die Parameter `row_number` und `column_number` festgelegt. Im Beispiel unten wäre `row_number == 4` und `column_number == 5`. (Erinnerung: Die Indizes von Listen starten bei 0.)



(Das Beispiel soll übrigens nur der besseren Veranschaulichung dienen. Unser Spielbrett in `grid` hat andere Dimensionen und enthält nur Werte `0`, `1` oder `2`.)

Um die einzelnen Werte der 4 Felder zu erhalten, gibt es die Funktionen `get_four_diagonal_up`, `get_four_diagonal_down`, `get_four_horizontal` und `get_four_vertical`.

Die Funktion `get_four_diagonal_up` ist im Code der Vorlage bereits richtig implementiert **und darf nicht mehr verändert werden**. Sie dient Ihnen aber als Vorlage für die verbleibenden Funktionen `get_four_diagonal_down`, `get_four_horizontal` und `get_four_vertical`. Im Moment greifen diese Funktionen noch alle auf die gleichen Positionen zu. Wie müssen die Indizes in `get_four_diagonal_down`, `get_four_horizontal` und `get_four_vertical` verändert werden, damit die Zugriffe stimmen wie in der Grafik oben abgebildet?

Hinweis: Falls `row_number` und `column_number` ein Feld festlegen, welches am Rand des Spielbretts liegt, können eventuell nicht alle 4 Richtungen geprüft werden. Deswegen sehen Sie im Code einen `try` - `except` -Block, welcher den passenden `IndexError` abfängt, sollte er auftreten. Sie müssen nichts mehr dafür tun.

Beispiele: Die Funktionen werden für `player_has_won` benötigt, sodass Sie darüber testen können, ob Ihre Veränderungen erfolgreich waren. Für `player_has_won(EXAMPLE_GRID_1)`, `player_has_won(EXAMPLE_GRID_2)` und `player_has_won(EXAMPLE_GRID_3)` wird `False` erwartet. Für `player_has_won(EXAMPLE_GRID_4)`, `player_has_won(EXAMPLE_GRID_5)`, `player_has_won(EXAMPLE_GRID_6)` und `player_has_won(EXAMPLE_GRID_7)` hingegen `True`. (Achten Sie darauf, mit den unveränderten Example-Grids zu testen. Eventuell haben Sie diese durch einen vorherigen Testaufruf von `drop_disc` verändert.)

Aufgabe 9 (1 Bonuspunkt)

Vervollständigen Sie die Funktion `next_player`. Wenn der übergebene Parameter `player` den Wert `1` enthält, soll `2` zurückgegeben werden. In **allen** anderen Fällen `1`.

Beispiele: Für `next_player(3)` oder `next_player(None)` erwarten wir als Rückgabewert `1`. Für `next_placer(1)` hingegen `2`.

Aufgabe 10 (1 Bonuspunkt)

Falls Sie alle anderen Aufgaben bereits erfolgreich bearbeitet haben, sollte das Spiel bereits funktionieren. Sie können das testen, indem Sie die Funktion `game_loop` aufrufen.

Bei der Benutzereingabe zur Auswahl der Spalte fehlt allerdings noch eine passende Fehlerbehandlung, für den Fall, dass der Benutzer keine Zahl zwischen 1 und 7 eintippt, sondern etwas anderes wie `42` oder `hello`.

Vervollständigen Sie die Funktion `player_choice`, sodass Fehleingaben nicht zum Abbruch des Programms führen. Tipp: Sie benötigen dafür im Regelfall sowohl einen `try`-`except`-Block, als auch mindestens eine `if`-Abfrage.
