Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Marius Weidner
Simon Dorer, Timpe Hörig

Universität Freiburg
Institut für Informatik
Wintersemester 2025

Übungsblatt 5

Abgabe: Montag, 17.11.2025, 9:00 Uhr

Geänderte Gesamtpunktzahl

Beachten Sie, dass ab diesem Übungsblatt – wie in der Einführungsveranstaltung angekündigt – die Gesamtpunktzahl pro Übungsblatt 20 Punkte beträgt. Die Anwesenheitspunkte für die Tutorien erhöhen sich ebenfalls auf 6 Punkte (ab nächster Woche).

Aufgabe 5.1 (Cocktails Mission; Datei: `overcooked.py`; Punkte: 5)

In dieser Aufgabe übernehmen Sie die Rolle eines Barkeepers, der Cocktails basierend auf einer begrenzten Auswahl an Zutaten zubereiten soll.

Gegeben sei eine Liste `recipes` mit Rezepten, wie zum Beispiel:

```
>>> recipes = [
...   ("Daiquiri", ["Rum", "Limette", "Zucker"]),
...   ("Mojito", ["Rum", "Limette", "Zucker", "Minze", "Soda"]),
...   ("Whiskey Sour", ["Whiskey", "Zitrone", "Zucker"]),
...   ("Tequila Sour", ["Tequila", "Zitrone", "Zucker"]),
...   ("Moscow Mule", ["Vodka", "Limette", "Ginger ale"]),
...   ("Munich Mule", ["Gin", "Limette", "Ginger ale"]),
...   ("Cuba Libre", ["Rum", "Coke"])
... ]
```

Schreiben Sie eine Funktion `mixable`, die eine Rezeptliste `recipes` (in der Form von oben) und eine Liste `ingredients` von verfügbaren Zutaten (`str`) als Argumente erhält. Die Funktion soll alle Rezepte aus `recipes` zurückgeben, die mit den angegebenen Zutaten zubereitet werden können.

Zum Beispiel:

```
>>> mixable(recipes, [])
[]
>>> mixable([], ["Orangensaft"])
[]
>>> mixable(recipes, ["Rum", "Whiskey", "Limette", "Zucker", "Coke", "Zitrone"])
['Daiquiri', 'Whiskey Sour', 'Cuba Libre']
>>> mixable(recipes, ["Rum", "Vodka", "Limette", "Zucker", "Ginger ale"])
['Daiquiri', 'Moscow Mule']
```

Aufgabe 5.2 (Kaprekar-Konstante; Datei: `kaprekar.py`; Punkte: 4)

Für alle ganzen Zahlen die (1) dreistellig sind (ggf. mit führenden Nullen) und (2) deren Ziffern nicht alle gleich sind gilt das Folgende:

Wenn Sie die Ziffern der Zahl in aufsteigender und absteigender Reihenfolge anordnen, die Differenz der beiden Zahlen bilden und diesen Vorgang wiederholen, dann erhalten Sie nach einigen Schritten immer die Zahl 495. Diese Zahl wird als Kaprekar Konstante bezeichnet.

Schreiben Sie eine Funktion `kaprekar`, die eine ganze Zahl `n` als Argument nimmt. Sie dürfen dabei davon ausgehen, dass `n` maximal eine dreistellige Zahl ist und nicht aus drei gleichen Ziffern besteht. Die Funktion soll die Anzahl der Schritte zurückgeben, die benötigt werden, um die Kaprekar Konstante 495 zu erreichen. Achten Sie darauf, dass Sie bei Zahlen mit weniger als drei Ziffern, führende Nullen hinzufügen müssen, damit die Zahl immer dreistellig ist.

Für die Eingabe `n = 42` ergibt sich zum Beispiel folgender Ablauf:

- $420 - 024 = 396$
- $963 - 369 = 594$
- $954 - 459 = 495$

Für `n = 42` benötigen wir also drei Schritte, um die Kaprekar Konstante zu erreichen.

Der Aufruf der Funktion soll also wie folgt aussehen:

```
>>> kaprekar(42)
3
>>> kaprekar(123)
5
>>> kaprekar(932)
3
>>> kaprekar(495)
0
>>> kaprekar(1)
6
```

Hinweis: Zum Sortieren der Zeichen einer Zeichenkette nach ihrem lexikographischen Wert können Sie die Python Funktion `sorted` verwenden. Diese sortiert die Zeichen in *aufsteigender* Reihenfolge und gibt eine Liste der sortierten Zeichen zurück.

```
>>> sorted("Hallo Welt")
[' ', 'H', 'W', 'a', 'e', 'l', 'l', 'l', 'o', 't']
>>> sorted("9287234")
['2', '2', '3', '4', '7', '8', '9']
```

Aufgabe 5.3 (Wordle; Datei: `wordle.py`; Punkte: 11)

In dieser Aufgabe programmieren Sie das beliebte Ratespiel *Wordle*. In unserer Version geht es darum ein Wort mit beliebiger Länge zu erraten, indem Sie aufeinander folgende Rateversuche eingeben. Das Programm gibt nach jedem Versuch Rückmeldungen, ob ein Buchstabe korrekt erraten wurde und sich an der richtigen Position befindet, ob er im gesuchten Wort vorkommt, aber an einer anderen Position, oder ob er gar nicht im Wort vorkommt.

Um die Implementierung etwas zu vereinfachen, verwenden wir nur Wörter, die keine doppelt vorkommenden Buchstaben enthalten.

Folgen Sie bei der Implementierung diesen Schritten:

(a) (4 Punkte) Schreiben Sie eine Funktion `next_guess`, die eine Wortlänge `word_length` als Argument nimmt, die Spielerin nach einer Eingabe fragt und diese zurück gibt. Wenn das eingegebene Wort die falsche Länge hat oder mehrfach vorkommende Buchstaben enthält, soll ein Fehler wie im Beispiel ausgeben werden und eine neue Eingabe gefordert werden.

Falls beide Bedingungen nicht erfüllt sind, soll nur die Fehlermeldung für die falsche Länge ausgegeben werden. Sobald ein gültiges Wort eingegeben wurde, gibt die Funktion das Wort in *Großbuchstaben* zurück.

Ein Aufruf der Funktion soll *exakt* so aussehen (Nutzereingaben sind blau, Ausgaben des Programms schwarz):

```
>>> guess = next_guess(6)
Nächster Tipp: pflaume
Ihr Wort hat nicht die geforderte Länge 6!
Nächster Tipp: banane
Ihr Wort enthält mehrfach vorkommende Buchstaben!
Nächster Tipp: kiwi
Ihr Wort hat nicht die geforderte Länge 6!
Nächster Tipp: kürbis
>>> guess
'KÜRBIS'
```

Hinweis: Sie können für diese Aufgabe die String-Methoden[1] `upper`[2] und `count`[3] verwenden:

```
>>> "banane".upper()
'BANANE'
>>> "BANANE".upper()
'BANANE'
```

[1] Methoden sind eine besondere Art von Funktionen. Im Wesentlichen wird beim Aufruf von `"banane".upper()`, die Funktion `upper` mit dem Argument `"banane"` aufgerufen. Wenn Sie fragen dazu haben, melden Sie sich gerne im Chat :)

[2] https://docs.python.org/3.12/library/stdtypes.html#str.upper

[3] https://docs.python.org/3.12/library/stdtypes.html#str.count

```
>>> "bAnAnE".upper()
'BANANE'
>>> "banane".count("n")
2
>>> "banane".count("b")
1
```

(b) (4 Punkte) Schreiben Sie eine Funktion `analyze_guess`, die ein geratenes Wort `guess` und das gesuchte Wort `solution` als Argumente nimmt. Die Funktion soll die Übereinstimmungen beider Wörter überprüfen und eine Zeichenkette zurückgeben, die den Status jedes Buchstabens im Rateversuch anzeigt:

- Großbuchstabe: Der Buchstabe ist korrekt und an der richtigen Position.
- Kleinbuchstabe: Der Buchstabe kommt im gesuchten Wort vor, befindet sich aber an einer anderen Position.
- Punkt (`"."`): Der Buchstabe kommt nicht im gesuchten Wort vor.

Sie können davon ausgehen, dass `guess` und `solution` gleich lang sind, keine doppelten Buchstaben enthalten und in Großbuchstaben vorliegen.

Zum Beispiel:

```
>>> analyze_guess("RAUM", "KAUM")
'.AUM'
>>> analyze_guess("MAUER", "LAMPE")
'mA.e.'
>>> analyze_guess("PAUSE", "OLIVE")
'....E'
>>> analyze_guess("WELT", "RAUM")
'....'
```

(c) (3 Punkte) Schreiben Sie nun eine Funktion `wordle`, die die beiden Funktionen aus den vorherigen Aufgabenteilen verwendet, um daraus ein richtiges Spiel zu machen. Die Funktion `wordle` soll dazu zunächst ein zufälliges Zielwort generieren und die Länge des Wortes ausgeben. Anschließend soll die Benutzerin so lange nach einem Wort gefragt werden, bis sie das Zielwort erraten hat. Wenn das gesuchte Wort erraten wurde, wird eine Erfolgsmeldung ausgegeben, und das Spiel endet. Wenn das Wort nur teilweise richtig ist, soll der String von `analyze_guess` ausgegeben werden.

Die Ausgabe des Spiels soll *exakt* wie in diesem Beispiel aussehen:

```
>>> wordle()
Das gesuchte Wort hat 5 Buchstaben!
Nächster Tipp: pause
> .A...
Nächster Tipp: pflaume
Ihr Wort hat nicht die geforderte Länge 5!
```

```
Nächster Tipp: lager
> .A..r
Nächster Tipp: traum
> tra..
Nächster Tipp: fahrt
Glückwunsch, Sie haben das Wort "FAHRT" erraten! :)
```

Hinweis: Zum Generieren der Zielwörter haben wir bereits eine Funktion `random_word` im Modul `words.py` via Git für euch bereitgestellt. Verwenden Sie diese in Ihrer Implementierung des Spiels:

```
>>> from words import random_word
>>> random_word()
'ALTER'
>>> random_word()
'BLICK'
>>> random_word()
'FLUCHT'
```

Aufgabe 5.4 (Erfahrungen; Datei: `NOTES.md`)

Notieren Sie Ihre Erfahrungen mit diesem Übungsblatt (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Editieren Sie hierzu die Datei `NOTES.md` im Abgabeordner dieses Übungsblattes auf unserer Webplattform. Halten Sie sich an das dort vorgegebene Format, da wir den Zeitbedarf mit einem Python-Skript automatisch statistisch auswerten. Die Zeitangabe `7.5 h` steht für 7 Stunden 30 Minuten.

Der Build-Server überprüft ebenfalls, ob Sie das Format korrekt angegeben haben. Prüfen Sie, ob der Build-Server mit Ihrer Abgabe zufrieden ist, so wie es im Video zur Lehrplattform gezeigt wurde.