

Programmierung für Naturwissenschaften 1
Wintersemester 2020/2021
Übungen zur Vorlesung: Ausgabe am 20.01.2021

Aufgabe 9.1 (1 Punkt) In der Datei `test_comprehensions.py` finden Sie die Implementierung einer Funktion `month_dict_get()`, die mit einer `return`-Anweisung ein Dictionary liefert, welches 12 Bezeichner für Monate auf die ganzen Zahlen von 0 bis 11 abbildet. Implementieren Sie in der Datei `comprehensions.py` eine Funktion `month_dict_get_comprehension()`, die mit einer *Dictionary comprehension* (siehe `python-slides.pdf`, Abschnitt *List comprehensions*, frame 16) das gleiche Dictionary zurückliefert, wie `month_dict_get()`. Für Ihre Implementierung kopieren Sie die Liste `month_list`. Außer dieser Liste darf Ihre Implementierung der genannten Funktion lediglich aus einer `return`-Anweisung mit der Dictionary comprehension bestehen, die genau eine Zeile umfasst.

In der Datei `test_comprehensions.py` finden Sie die Implementierung einer Funktion `partial_sum_gen(g)`, die die Partialsumme einer Folge von Zahlen liefert, die durch `g` generiert wird. Falls die Zahlen a_0, a_1, a_2, \dots von `g` generiert werden, dann generiert `partial_sum_gen(g)` die Zahlen $a_0, a_0 + a_1, a_0 + a_1 + a_2, \dots$.

Implementieren Sie in der Datei `comprehensions.py` eine Funktion

`partial_sum_gen_comprehension(g)`,

die mit einer *Generator comprehension* (siehe `python-slides.pdf`, Abschnitt *List comprehensions*, frame 12) die gleiche Funktionalität erreicht wie `partial_sum_gen`. Ihre Implementierung darf (nach dem Funktionskopf) nur genau eine Zeile enthalten, also eine `return`-Anweisung mit der generator comprehension.

Durch `make test` verifizieren Sie, dass Ihre Implementierungen der beiden Funktionen für die verwendeten Testdaten korrekt funktionieren.

Punkteverteilung: je 0.5 Punkte für die beiden Funktionen.

Aufgabe 9.2 (4 Punkte) In dieser Aufgabe soll auf Basis der Funktionen aus Aufgabe 8.2 ein lauffähiges Programm `pwgen.py` zum Generieren von Passwörtern implementiert werden. In den Materialien zu dieser Aufgabe finden Sie die Datei `pwgen_template.py`, die Sie umbenennen in `pwgen.py`. In der Datei `pwgen_functions.py` finden Sie die Musterlösung zu Aufgabe 8.2, die Sie gerne nutzen dürfen.

Zunächst implementieren Sie die Funktion `password_generate(wd, struct_str)`, die entsprechend des Dictionaries `wd` (berechnet durch `word_dict_get()`) und des Strukturstrings `struct_str` ein Paar `(pw, choice_list)` zurückliefert. Dabei ist `pw` ein zufällig generiertes Passwort und `choice_list` ist die Liste der Anzahlen der möglichen Passwörter für die einzelnen Strukturelemente in `struct_str`.

Zum Aufzählen der Strukturelemente verwenden Sie die Funktion `structure_elements_enumerate` aus Aufgabe 8.2. Für die Generierung der Teile des Passwortes entsprechend eines d- bzw. p-Strukturelements der Länge `n` verwenden Sie den Ausdruck `randstring(string.digits, n)`

bzw. `randstring(string.punctuation, n)`. Dabei ist `randstring` die in Aufgabe 8.2 entwickelte Funktion. Hierfür müssen Sie das Modul `string` importieren. Für die Generierung der Teile des Passwortes mit einem `w`-Strukturelement der Länge `m` wählen Sie ein Wort aus der Liste `wd[m]` zufällig aus. Falls `m` kein Schlüssel in `wd` ist (also kein Wort der Länge `m` in `wordlist.txt` existiert), muss eine Fehlermeldung generiert und das Programm mit dem Exitcode 1 abgebrochen werden.

Die einzelnen Werte in `choice_list` ergeben sich wie folgt:

- bei einem `d`- oder `p`-Strukturelement berechnet man den Wert aus der Anzahl der Zeichen im entsprechenden Alphabet und der Länge `m`.
- bei einem `w`-Strukturelement ist der Wert die Anzahl der Worte in `wd[m]`

Beispiel: Für den Strukturstring `w4p2d2w5` mit 4 Strukturelementen ist `choice_list` die Liste `[2143, 1024, 100, 3132]`.

Im zweiten Schritt implementieren Sie eine Funktion `parse_command_line()`, die unter Nutzung des Moduls `argparse` einen Argumentparser `p` mit den folgenden Optionen spezifiziert und `p.parse_args()` zurückliefert. Siehe auch `python-slides.pdf`, Abschnitt *Functions*, frame 58-64.

- eine Option `-s/--structure` mit genau einem String-Argument, nämlich ein Strukturstring, wie oben beschrieben. Der default-Wert ist `'w4p2d2w5p1d1w8'` und dieser soll bei der Spezifikation dieser Option angegeben werden.
- eine Option `-n/--number` mit genau einem ganzzahligen Argument, das die Anzahl der auszugebenden Passwörter spezifiziert. Der default-Wert ist 1 und dieser bei der Spezifikation dieser Option definiert werden.

Die Funktion `main()`, die das Hauptprogramm implementiert und ihr Aufruf sind bereits in der vorgegebenen Datei vorhanden. Durch `make test` verifizieren Sie, dass Ihr Programm für einige Testdaten korrekt funktioniert.

Punkteverteilung:

- 2 Punkte für die Implementierung der Funktion `password_generate`.
- 1 Punkt für die Implementierung von `parse_command_line()`.
- 1 Punkt für die bestandenen Tests.

Aufgabe 93 (5 Punkte) In dieser Aufgabe geht es darum, in Python3 eine Klasse `Morse` zur Codierung und Decodierung eines Textes durch Morse-Zeichen zu implementieren. Für jedes alphanumerische Zeichen sowie die Zeichen `.` und `,` ist der Morse-Code eine Folge zweier Signallängen (kurz und lang, dit und dah im Englischen). Diese Signallängen werden durch die Zeichen `.` und `-` beschrieben. Einen String, der nur aus den Zeichen `.` und `-` besteht, nennen wir *Morse-String*.

In der Datei `morseClass_template.py` finden Sie eine Basis-Implementierung der Klasse `Morse` mit einem Dictionary `morse_code`, das die Codierung der genannten Zeichen in einen Morse-String definiert. `morse_code` enthält zusätzlich eine Codierung für das Leerzeichen. Benennen Sie die Datei `morseClass_template.py` um in `morseClass.py`.

Ihre Aufgabe besteht aus den folgenden Teilaufgaben.

1. Implementieren Sie in der Klasse `Morse` eine Methode `encode(self, text)`, die einen String `text` als Argument erhält und mit einer `return`-Anweisung den entsprechenden

Morse-String zurückliefert. Bei der Codierung sollen Kleinbuchstaben wie die entsprechenden Großbuchstaben behandelt werden. Während bei der traditionellen Anwendung von Morse-Codes zwischen der Codierung von zwei aufeinanderfolgenden Zeichen eine kurze Pause erfolgt, die man in einem String typischerweise durch ein Leerzeichen codiert, soll das in Ihrer Implementierung nicht erfolgen. Beispiel: Für den String SOS liefert die Funktion den Morse-String `...---...`

1 Punkt

2. Wenn das Programm `morseClass.py` mit der Option `--text` aufgerufen wird, wird der Morse-String angezeigt. Im Makefile sind einige Tests implementiert, die testen, ob Ihre Funktion `encode` korrekt funktioniert.
3. Wenn das Programm `morseClass.py` nicht mit der Option `--text` oder `--decode` aufgerufen wird, dann wird ein Shell-Skript generiert, das entsprechend der Zeichen des Morse-Strings ein Programm zum Abspielen der Dateien `dit.wav` und `dah.wav` aufruft. Wenn man dieses Shell-Skript über eine Pipe mit dem Befehl `sh -s` verbindet (siehe Makefile), kann man den Morse-String hören. Dafür muss unter macOS das Programm `afplay` und unter Linux das Programm `aplay` verfügbar sein. Wenn das bei Ihnen der Fall ist, können Sie sich die Morsezeichen anhören (falls der Lautsprecher an ist). Testen Sie die Funktionalität durch den Aufruf von `make test_sound`.¹

4. Auch wenn man den Morse-Code kennt, kann man den durch die Funktion `encode` gelieferten Morse-String nicht immer eindeutig decodieren. Geben Sie eine Erklärung hierfür anhand eines Beispiels, an dem Sie das Problem verdeutlichen. Zu Beantwortung dieser Fragen müssen Sie sich in `morseClass.py` das Dictionary `morse_code` ansehen.

1 Punkt

5. Um eine eindeutige Decodierung eines Morse-Strings zu ermöglichen, muss man eine andere Codierung verwenden, wie z.B. die Codierung entsprechend dem Dictionary `morse_code2_0`. Wenn man die Option `--mc2_0` wählt, erfolgt die Codierung entsprechend diesem Dictionary. Beispiel: Der Code von SOS entsprechend `morse_code2_0` ist `..-.-.....-`.

6. Implementieren Sie in der Klasse `MorseClass` eine Methode `decode`, die einen Morse-String (entsprechend der Codierung mit `morse_code2_0`) als Argument erhält und den hierdurch codierten Text mit einer `return`-Anweisung als Ergebnis liefert. Beispiel: Für den obigen Morse-String `..-.-.....-` liefert diese Funktion den decodierten String SOS.

2 Punkte

7. In den Materialien finden Sie zwei Dateien `unknown_short.code2_0` und `unknown.code2_0` mit Morsestrings (Version 2.0) für zwei unbekannte Texte. Die Originaltexte sind nicht Teil der Testdaten. Trotzdem kann durch den Aufruf von `make test_decode` getestet werden, ob Ihre Implementierung der Methode `decode` korrekt funktioniert. Beschreiben Sie, warum das möglich ist. Dafür müssen Sie sich das Makefile und den Optionsparser ansehen und ausprobieren, was die einzelnen Kommandos beim Aufruf von `make test_decode` bewirken. Schauen Sie sich außerdem den Text an, der durch `./morseClass.py -d unknown.code2_0` ausgegeben wird. Der Text erläutert eine wichtige Eigenschaft von Codierungen.

1 Punkt

Für die Aufgabenteile, die nicht Teil der Implementierung sind, schreiben Sie bitte Text in Form eines Kommentars an der Ende der Datei. Ein Text mit mehreren Zeilen wird durch `'''` am Anfang

¹Leider funktioniert dieser Test nicht in der virtuellen Linux-Version unter MS-Windows. Aber man kann sich dann einfach die Datei `SOS_sound.mp4` direkt unter MS-Windows anhören oder `make --MS_WINDOWS SOS` aufrufen und die im Terminal angezeigte Folge von Anweisungen unter MS-Windows ausführen. Diese Lösung wurde von Time Wacke vorgeschlagen.

und am Ende des Textes begrenzt.

Bitte die Lösungen zu diesen Aufgaben bis zum 25.01.2021 um 18:00 Uhr an pfn1@zbh.uni-hamburg.de schicken.