Prof. Dr. Stefan Göller
Dr. Da-Jung Cho
Daniel Kernberger

Einführung in die Informatik

WS 2019/2020

Übungsblatt 7
06.12.2019 - 12.12.2019

Abgabe: Bis zum 12.12. 18:00 Uhr über moodle. Reichen Sie pro Aufgabe, die Sie bearbeitet haben, genau eine Textdatei mit dem Namen `aufgabe_i.py`, (wobei `i` die Aufgabennummer ist) ein, welche die Lösung ihrer Gruppe enthält. Verwenden Sie die Vorlagen, die wir in Moodle hinterlegt haben.

Beachten Sie bitte folgende Hinweise zur Abgabe. Verwenden Sie keine globale Variablen, löschen Sie alle Ihre `print`-Anweisungen (außer natürlich diejenigen, die wir selbst in den Templates bereitstellen!) und fügen Sie möglichen Testcode in den Bereich ein, den wir in den Templates dafür vorgeben, nämlich nach `if __name__ == "__main__"`. Wenn diese obigen Vorgaben nicht eingehalten werden, werden ggf. Punkte abgezogen.

Aufgabe 1 (Glass-Box-Test) (20 Punkte):

Gegeben sei die Funktion `next-palindrome(num, k)`, die einen String `num` (der nur aus Buchstaben aus $\{0, 1, \ldots, 9\}$ besteht) und einen positiven Integer `k` entgegennimmt. Die Funktion berechnet die nächsten `k` Zeichenketten, welche als Integer interpretiert, mindestens so groß wie `num` (als Integer interpretiert) ist und ein Palindrome ist. Zur Erinnerung, ein Palindrom ist eine Zeichenkette, die sich vorwärts wie rückwärts gleich liest. Falls Ihre übergebenen Parameter bspw. `num`=`387654` und `k`=6 wäre, dann gäbe `next-palindrome` die Zeichenketten `387783, 388883, 389983, 390093, 391193, 392293` aus.

```
def next_palindrome(num,k):
    while k > 0:
        palindrome = True
        for i in range(len(num) // 2):
            if num[i] != num[-1 -i]:
                palindrome = False
                break

        if palindrome:
            k -= 1
            print('palindrome:' + num)
            num = str(int(num) + 1)
        else:
            num = str(int(num) + 1)
if __name__ == "__main__":
    # Fuehren Sie hier den Glass Box Test fuer Funktion next_palindrome durch

    next_palindrome("23",3) ## schreiben Sie hier was dieser Aufruf abdeckt
```

Führen Sie den Glass-Box-Test für die Funktion `next_palindrome` durch, indem Sie im Test-Bereich weitere Testfälle angeben, die einem Glass-Box-Test entsprechen. Kommentieren Sie welche Aufrufe was genau abdecken.

Beispiel für einen Glass-Box-Test: Gegeben sei folgende Funktion `add(n,m)`.

```
def addiere(n,m):
    while n > 0:
        if m > 100:
            print(m)
            break
        else:
            m = m + n
            n = n-1
```

Folgende Aufrufe (Eingabeparameterpaare) lieferten einen Glass-Box-Test für die Funktion `addiere`.

- Der Aufruf `add(0, 20)` führt die `while`-Schleife nicht aus; `add(1,99)` führt die `while`-Schleife genau ein Mal aus; `add(3,12)` führt die `while`-Schleife mehr als einmal aus, nämlich drei Mal. Durch diese drei Aufrufe wird die `while`-Schleife also überdeckt.
- Der Aufruf `add(2, 102)` überdeckt den `if m > 100` - Zweig, also auch die Zeile `print(m)` und die `break`-Anweisung.

- Der Aufruf `add(20,39)` überdeckt den `else`-Zweig als auch die Zuweisungen `m = m + n` und `n = n-1`.

Aufgabe 2 (Zyklische Integer-Wörterbucher) (20 Punkte):

Schreiben Sie eine Funktion `zyklisch(wb,i)`, die zu Beginn folgende Assertions bereitsstellt:

- Durch eine erste Assertion soll sichergestellt werden, dass es sich beim Parameter `wb` um ein Wörterbuch handelt, mit der Fehlermeldung 'erster Parameter kein Woerterbuch'.
- Durch eine zweite Assertion soll sichergestellt werden dass es sich beim Parameter `i` um einen Integer handelt, mit der Fehlermeldung 'zweiter Parameter kein Integer'.
- Durch eine dritte Assertion soll sichergestellt werden, dass sowohl alle Schlüssel als auch alle Werte im Wörterbuch `wb` vom Typ `int` sind, mit der Fehlermeldung 'erster Parameter kein int-int Woerterbuch'.
- Durch eine vierte Assertion soll sichergestellt werden, dass der zweite Parameter `i` tatsächlich ein Schlüssel im übergebenen Wörterbuch ist, mit der Fehlermeldung 'zweiter Parameter kein Schlüssel des ersten Parameters'.

Danach soll überprüft werden, ob durch wiederholtes Nachschlagen im Wörterbuch, beginnend ab dem Startwert `i` ein Lasso entsteht. Was mit einem *Lasso* gemeint ist, soll anhand dreier Beispiele klargemacht werden:

- Wenn `wb` = {`1:2,2:3,3:4,4:2`} und `i` = 1 entsteht ein Lasso, denn durch erstmaliges Anwenden des Wörterbuchs auf den Startschlüsselwert `i` mit dem Wert 1 (notwendigerweise als Schlüssel im Wörterbuch vorhanden, denn sonst wäre die obige vierte Assertion nicht sichergestellt), erhalten wir den Wert `2`. Der Wert `2` ist als Schlüssel im Wörterbuch vorhanden und wird auf den Wert `3` abgebildet. Die `3` ist auch als Schlüssel in `wb` vorhanden und wird auf den Wert `4` abgebildet. Auch die `4` ist ein Schlüssel im Wörterbuch und wird auf `2` abgebildet. Wir erhalten also folgendes Lasso:

  $$\texttt{1} \to \texttt{2} \to \texttt{3} \to \texttt{4} \to \texttt{2}$$

  Der Schlüsselwert `2` ist also die erste Position ab der Kreis zu sich selbst existiert.

- Ebenso erhielten wir das Lasso

  $$\texttt{14} \to \texttt{2} \to \texttt{15} \to \texttt{14}$$

  für `wb` = {`14:2,19:3,2:15,15:14,3:3`} und `i` = 14. Ein Lasso kann also auch selbst ein Kreis sein.

- Bei Eingabe `wb` = {`11:2,19:3,2:15,15:100,3:3`} und `i` = 11 erhielten wir kein Lasso, denn der "Pfad ab `i`" lautet

$$\texttt{11} \to \texttt{2} \to \texttt{15} \to 100$$

und endet in einer Sackgasse `100`, da `100` nicht als Schlüssel in `wb` vorkommt.

Aufgabe 3 (Arithmetik auf Brüchen) (20 Punkte):

Ein *Bruch* ist für uns ein Paar (`a`, `b`), wobei `a` und `b` jeweils vom Typ `int` sind und `b` > 0 gilt. Ein Bruch lautet *gekürzt*, falls entweder (`a` $= 0$ und `b` $= 1$) oder `a` $\neq 0$ und ggT($|$`a`$|$, `b`) $= 1$, wobei $|\cdot|$ Betragsoperation bezeichnet.

Schreiben Sie zunächst eine Funktion `istBruch(x)`, die überprüft, ob es sich beim Argument `x` tatsächlich um einen Bruch handelt.

Schreiben dann folgende Funktionen.

- Eine Funktion `kuerze`(`x`), die den übergebenen Bruch kürzt. Verwenden Sie den euklischen Algorithmus, den wir Ihnen im Template `aufgabe_3.py` hierfür zur Verfügung stellen.
- Eine Funktion `plus`(`x`, `y`), die die Summe der Brüche `x` und `y` berechnet. Das Ergebnis soll gekürzt zurückgegeben werden.
- Eine Funktion `minus`(`x`, `y`), die die Differenz der Brüche `x` und `y` berechnet. Das Ergebnis soll gekürzt zurückgegeben werden.
- Eine Funktion `mal`(`x`, `y`), die das Produkt der Brüche `x` und `y` berechnet. Das Ergebnis soll gekürzt zurückgegeben werden.
- Eine Funktion `teile`(`x`, `y`), die den Quotienten der Brüche `x` und `y` berechnet. Falls `y` den Wert 0 darstellt, soll eine `ZeroDivisionError`-Exception geworfen werden. Das Ergebnis soll gekürzt zurückgegeben werden.

Dabei sollen alle obigen Funktionen außer `istBruch(x)` zu Beginn mittels einer Assertion sicherstellen dass es sich bei beiden Argumente (bzw. dem Argument) tatsächlich um Brüche (bzw. um einen Bruch) handelt — Fehlermeldung: 'Leider handelt es sich nicht um zwei Brüche' (bzw. 'Leider handelt es sich um keinen Bruch').