

Programmierung für Naturwissenschaften 1
Wintersemester 2020/2020
Übungen zur Vorlesung: Ausgabe am 02.12.2020

Aufgabe 4.1 (2 Punkte) In der Vorlesung haben Sie `while`- und `for`-Schleifen kennengelernt.

In den Materialien zur Übung finden Sie eine Datei `loops_orig.py`. Diese enthält 2 `for`-Schleifen und 2 `while`-Schleifen. Erstellen Sie eine Kopie der Datei mit dem Namen `loops.py`. Ersetzen Sie darin alle `for`-Schleifen durch `while`-Schleifen und umgekehrt. Bei den `while`-Schleifen darf nicht der `range`-Operator verwendet werden.

In den Materialien zur Übung finden Sie eine Testdatei und ein `Makefile`. Darin ist ein Test implementiert, der Ihr Programm aufruft und das Ergebnis mit Hilfe des Linux-Tools `diff` mit dem erwarteten Ergebnis vergleicht. Diesen Test können Sie mit dem Befehl `make test` in der Linux Shell ausführen (Achtung: Ein erfolgreicher Test bedeutet nicht unbedingt, dass die Aufgabe korrekt gelöst ist. Die Aufgabe gilt selbstverständlich erst dann als gelöst, wenn die Umwandlung bzgl. der Form der Schleifen auch durchgeführt wurde.)

Punktevergabe: Je 0.5 Punkte für die korrekte Konvertierung der Schleifentypen.

Aufgabe 4.2 (5 Punkte) Schreiben Sie ein Python-Skript `zahlenreihen.py`, das für eine positive ganze Zahl k die folgenden Zahlenreihen und jeweils ihre Summe zeilenweise ausgibt:

- a) $2, 4, 6, \dots, 2k$
- b) $\frac{5}{2}, \frac{7}{4}, \frac{9}{8}, \frac{11}{16}, \dots, \frac{3+2k}{2^k}$
- c) $1, -\frac{1}{2}, \frac{1}{4}, -\frac{1}{8}, \dots, \frac{(-1)^{k-1}}{2^{k-1}}$
- d) $\frac{1}{1!}, \frac{1}{2!}, \frac{1}{3!}, \frac{1}{4!}, \dots, \frac{1}{k!}$

Hinweise:

- Für die Berechnung der Zahlenreihen sollen `for`-Schleifen und die Methode `range` verwendet werden.
- Es dürfen jeweils nur die Grundrechenarten `+`, `*`, `-` und `/` verwendet werden, jedoch nicht der Operator `**` und keine Funktion oder eine eigene Schleife zur Berechnung der Potenz oder der Fakultät für jedes Reihenelement. Sie müssen sich also überlegen, wie jeweils ein Reihenelement aus dem Vorherigen durch Verwendung der Grundrechenarten berechnet wird.
- Die Übergabe von k erfolgt über die Kommandozeile, d.h. über die Liste `sys.argv`. Sie müssen daher überprüfen, ob `sys.argv` die passende Länge 2 hat. Ist das nicht der Fall, dann soll die Zeile

Usage: `./zahlenreihen.py <k>`

mit `sys.stderr.write()` ausgegeben werden.

- Die Kommandozeilenparameter in `sys.argv` sind Strings. Daher muss der String in `sys.argv[1]` mit der Methode `int()` in eine ganze Zahl umgewandelt werden. Damit das Programm bei ungültigen Eingaben mit einer verständlichen Fehlermeldung abbricht, muss mit `try/except` eine Ausnahmebehandlung erfolgen, die in etwa so aussehen kann:

```

try:
    k = int(sys.argv[1])
except ValueError as err:
    sys.stderr.write(formatstring.format(sys.argv[0], sys.argv[1]))
    exit(1)

```

Dabei ist `formatstring` ein String mit zwei Platzhaltern, so dass beim Aufruf von

`./zahlenreihen.py abc`

die folgende Fehlermeldung ausgegeben wird:

`./zahlenreihen.py: cannot convert 'abc' to int`

- Falls der String in eine Zahl k konvertiert wurde, muss noch geprüft werden, dass es sich bei k um eine positive Zahl handelt. D.h. bei Eingabe eines Wertes ≤ 0 soll eine Fehlermeldung ausgegeben werden. Z.B. soll beim Aufruf `./zahlenreihen.py -3` die Fehlermeldung `./zahlenreihen.py: parameter -3 is not positive int` ausgegeben werden.
- Fehlermeldungen werden nach `sys.stderr` ausgegeben und führen zu einem Abbruch des Programms mit `exit(1)`.
- In den Materialien finden Sie eine Datei mit der erwarteten Ausgabe des Programms für $k = 10$. Dabei werden die reellen Zahlen, die nicht ganzzahlig sind, in wissenschaftlicher Notation angegeben. Verwenden Sie als Platzhalter im Formatstring für die Ausgabe dieser Zahlen `{:.5e}`. Durch `make test10` wird die Ausgabe Ihres Programms mit dieser Datei verglichen. Durch `make test_err` verifizieren Sie, dass Sie Fehlerfälle entsprechend der obigen Spezifikation korrekt behandelt haben. Durch `make test` werden beide Teiltests durchgeführt.

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Flow of control*, frame 29-32, 36-38

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Histograms: counting occurrences of values*, frame 3

Punktevergabe:

- je 1 Punkt für die korrekt berechneten Zahlenreihen.
- 1 Punkt für die korrekte Behandlung der Fehler.

Aufgabe 4.3 (3 Punkte) Die Quersumme einer ganzen Zahl ist die Summe der Ziffern, aus der diese Zahl besteht. Schreiben Sie ein Python-Skript `quersumme.py`, das die Quersumme einer beliebigen negativen oder positiven ganzen Zahl berechnet und ausgibt. Einer positiven ganzen Zahl kann optional das Zeichen `+` vorangestellt werden. Sie können voraussetzen, dass das Zeichen `+` und `-` (falls es verwendet wird) jeweils direkt vor den Ziffern vorkommt.

Die Zahl soll als String auf der Kommandozeile übergeben werden und dieser String enthält möglicherweise am linken und rechten Rand Leerzeichen. Für den Fall, dass das Skript nicht mit der korrekten Anzahl von Argumenten aufgerufen wird, soll die Fehlermeldung

Usage: `./quersumme.py <integer>`

auf `sys.stderr` ausgegeben werden.

Überprüfen Sie mit Hilfe eines regulären Ausdrucks die Eingabe auf Korrektheit und geben Sie eine Fehlermeldung auf `sys.stderr` aus, falls das nicht so ist. Der Aufruf `./quersumme.py 1.5` soll die Fehlermeldung

```
./quersumme.py: argument "1.5" is not an integer
```

auf `sys.stderr` liefern. Bei Fehlermeldungen bricht das Programm mit `exit(1)` ab. Ihr Programm darf die Funktion `int` zur Konvertierung von Strings in ganze Zahlen nicht verwenden. Zur Umwandlung von Ziffern in die entsprechenden ganzen Zahlen verwenden Sie die Methode `ord`, die für einen String `ds`, der nur aus einer Ziffer besteht, jeweils die Werte entsprechend der folgenden Tabelle liefert:

<code>ds</code>	<code>'0'</code>	<code>'1'</code>	<code>'2'</code>	<code>'3'</code>	<code>'4'</code>	<code>'5'</code>	<code>'6'</code>	<code>'7'</code>	<code>'8'</code>	<code>'9'</code>
<code>ord(ds)</code>	48	49	50	51	52	53	54	55	56	57

In den Materialien finden Sie eine Dateien mit Aufrufen des zu entwickelnden Programms und den erwarteten Ergebnissen. Durch `make test_positive` wird die Ausgabe Ihres Programms mit den erwarteten Ergebnissen verglichen. Durch `make test_err` verifizieren Sie, dass Sie Fehlerfälle entsprechend der obigen Spezifikation korrekt behandelt haben. Durch `make test` werden beide Teiltests durchgeführt.

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Regular expressions*, frame 5

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Histograms: counting occurrences of values*, frame 10

Punktevergabe:

- 1 Punkt für den regulären Ausdruck zu Fehlerbehandlung
- 2 Punkte für die korrekte Konvertierung der Strings in ganze Zahlen (inklusive erfolgreicher Tests)

Bitte die Lösungen zu diesen Aufgaben bis zum 07.12.2020 um 18:00 Uhr an `pfn1@zbh.uni-hamburg.de` schicken.

Hinweise zur Nutzung der Umgebungsvariable `PATH`:

Wenn man unter Unix ein Programm `p` ausführen möchte, wird der Installationsort (Pfad) anhand der Umgebungsvariablen `PATH` ermittelt. Es werden alle in der Umgebungsvariablen `PATH` gelisteten Verzeichnisse (Ausgabe mittels `echo $PATH`) durchsucht. Sei `d` der erste Pfad, der eine ausführbare Version des Programms `p` enthält. Dann wird das Programm `p` im Verzeichnis `d`, notiert durch `d/p` ausgeführt. Wenn man ein Programm aufrufen möchte, das in keinem der in `PATH` spezifizierten Pfade vorhanden ist, so muss der entsprechende Pfad dem Programm vorangestellt werden.

Durch

```
export <Variable>=<Wert>
```

lassen sich in der `bash` oder `zsh` Werte von Umgebungsvariablen wie `PATH` festlegen. Nutzen Sie `export` um in der Datei `.bashrc` (falls Sie `bash` verwenden) oder `.zsh` (falls Sie `zsh` verwenden) die Pfadliste in `PATH` zu erweitern:

```
export PATH=${PATH}:${HOME}/pfn1_2019/bin
```

Das setzt voraus, dass das geklonte Repository in Ihrem `HOME`-Verzeichnis liegt. Falls das nicht der Fall ist, muss nach dem Doppelpunkt der Pfad entsprechend angepasst werden.

Nachdem Sie obige Zeilen in `.bashrc` eingetragen haben, muss noch im Terminal

```
. ~/.bashrc
```

ausgeführt (oder `z` statt `ba` für `zsh`).

Beim Erweitern der Variablen `PATH` ist zu beachten, dass die bereits in `PATH` definierten Pfade erhalten bleiben. Beachten Sie die Verwendung des Paares von geschweiften Klammern.

Die Reihenfolge der Verzeichnisse in der Variablen `PATH` ist dabei wichtig, wenn es mehrere Verzeichnisse mit gleichen Programmnamen gibt: nur das erste Verzeichnis in der Pfadliste, das das Programm enthält, wird berücksichtigt.

Hinweise zur Anfertigung Ihrer Lösungen

Bitte beachten Sie die Hinweise zur Anfertigung von Lösungen der Übungsaufgaben, die Ihnen ausgehändigt wurden. Achten Sie insbesondere auf Folgendes:

- Erfolgreiche Tests durch Aufruf von `make test`. Wenn ein Test nicht erfolgreich ist, dann muss das dokumentiert werden.
- Zeilenlänge ≤ 80 in den Python-Dateien. Das Skript `pfn1_2020/bin/code_check.py`, angewendet auf beliebig viele Dateinamen, sucht nach Zeilen, die länger als 80 Zeichen lang sind und liefert die Zeilennummer der ersten solchen Zeile.
- Damit Sie Ihr Programm auf 80 Zeichen pro Zeile formatieren können, müssen Sie es ggf. an den passenden Stellen umbrechen. Z.B. können die Parameter einer Funktion in verschiedenen Zeilen stehen, da sie durch ein Paar von Klammern begrenzt werden. Dabei sollten Sie darauf achten, dass Parameter, die in der nächsten Zeile stehen, rechts von der öffnenden Klammer steht. Dadurch wird der Programmcode besser lesbar. Längere Stringlitterale zerlegen Sie an Wortgrenzen in einzelne Strings, die mit einer runden Klammer zusammengehalten werden. Wenn mehrere Zeilen logisch zusammen gehören, kann man sie durch ein `\` am Ende der Zeile trennen, siehe z.B. section 8, frame 7 in `python-slides.pdf`.

Wenn Sie die obigen Schritte zur Ergänzung der Umgebungsvariable `PATH` durchgeführt haben, dann können Sie `code_check.py` ohne Angabe des Pfades aufrufen.