Programmierung I
Wintersemester 2021/2022
Markus Menth, Martin Gropp

DHBW
Duale Hochschule
Baden-Württemberg

Aufgabenblatt 3 vom 17. November 2021, Abgabe am 3. Dezember 2021, 22:00 Uhr

---

Aufgabe 3.1: Grundlagenübung Rekursion ≙

14 Punkte
Rekursion

Legen Sie eine Datei `grundlagen.py` an und lösen Sie darin die folgenden Aufgaben.
Verwenden Sie Kommentare, um zu kennzeichnen, zu welcher Aufgabe eine Lösung gehört.

1. Implementieren eine rekursive Funktion `def divide_int(a: int, b: int) -> int`, welche die ganzzahlige Division zweier Zahlen $a \in \mathbb{N}$ und $b \in \mathbb{N}$ ausführt. Die Funktion soll $\lfloor \frac{a}{b} \rfloor$, also das Ergebnis der Division auf die nächste ganze Zahl abgerundet, zurückgeben. Verwenden Sie ausschließlich Additionen und/oder Subtraktionen, keine Multiplikationen oder Divisionen!

2. Die Potenzierung a^b lässt sich für $b \in \mathbb{N}_0$ auf eine wiederholte Multiplikation $(a \cdot a \cdot \ldots \cdot a)$ zurückführen.

   Finden Sie eine entsprechende rekursive Definition und implementieren Sie damit die Funktion `def pow(a: int, b: int) -> int`, welche als Ergebnis den Wert a^b zurückgibt.

   Sie können davon ausgehen, dass $a \in \mathbb{Z}$ und $b \in \mathbb{N}_0$ sind.

3. Schreiben Sie eine rekursive Funktion `def countdown(n: int) -> None`, die die Zahlen von `n` bis 0 in absteigender Reihenfolge auf die Standardausgabe ausgibt. Überlegen Sie sich:

   - Welche Zahl muss in `countdown(n)` ausgegeben werden?
   - Was muss passieren, wenn man `countdown(n-1)` aufruft?
   - Was ist eine geeignete Abbruchbedingung?

   Programmieren Sie nun die Funktion und testen Sie sie!

4. Wenn die Funktion `countdown` richtig funktioniert, kopieren Sie sie und benennen Sie sie um in `countup`. Ändern Sie auch den rekursiven Aufruf entsprechend.

   Verschieben Sie nun lediglich eine Zeile an eine andere Stelle, so dass ein Aufruf von `countup(n)` die Zahlen von 0 bis `n` in *aufsteigender* Reihenfolge ausgibt!

5. Die *Leibniz-Reihe* ist eine von Gottfried Wilhelm Leibniz entwickelte Formel zur Berechnung der Kreiszahl π. Sie lautet:

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} = \cdots = \frac{\pi}{4}$$

   Schreiben Sie eine rekursive Funktion `def approx_pi(n: int)`, die eine Näherung für π mit Hilfe der *Leibniz-Reihe* bis $k =$ `n` berechnet und zurückgibt. Sie können für die Berechnung von $(-1)^k$ im Zähler Ihre soeben geschriebene Funktion `pow` benutzen.

   Vorsicht: Die Reihe hat den Wert $\frac{\pi}{4}$!

6. Gegeben ist folgende rekursive Funktion `f`:

```
def f(x: int, y: int) -> int:
  if x == y:
    return x

  return f((x + y) // 2, x) + x
```

   Programmieren Sie eine iterative Version dieser Funktion! Die neue Funktion soll ebenfalls den Namen `f` tragen und die gleichen Parameter akzeptieren.

7. Schreiben Sie eine rekursive Funktion `def switch_case(s: str) -> str`. Diese soll einen übergebenen String `s` Zeichen für Zeichen so umwandeln, dass alle Kleinbuchstaben zu Großbuchstaben werden und umgekehrt, und den umgewandelten String zurückgeben. Dies soll für alle Buchstaben a–z bzw. A–Z funktionieren. Umlaute sowie Satz- und andere Sonderzeichen sollen unverändert bleiben.

   Die vordefinierten Funktionen `ord` und `chr` helfen Ihnen, Zeichen aus einem String in Zahlen umzuwandeln und umgekehrt. Informieren Sie sich darüber in der Python-Dokumentation! Den Zeichen `A` bis `Z` sind aufsteigende Zahlencodes direkt hintereinander zugeordnet, ebenso den Zeichen `a` bis `z`. In Unicode (oder auch ASCII) sind das: 65 (A) – 90 (Z) bzw. 97 (a) – 122 (z).

   Die Methoden `upper`, `lower`, `islower`, `isupper` u. ä. sollen nicht verwendet werden!

   Beispiel für die Rückgabe des Aufrufes `switch_case('Programmierung␣1␣WiSe␣2021/22')`:

```
pROGRAMMIERUNG 1 wIsE 2021/22
```

Aufgabe 3.2: Binäre Suche 10 Punkte

Rekursion

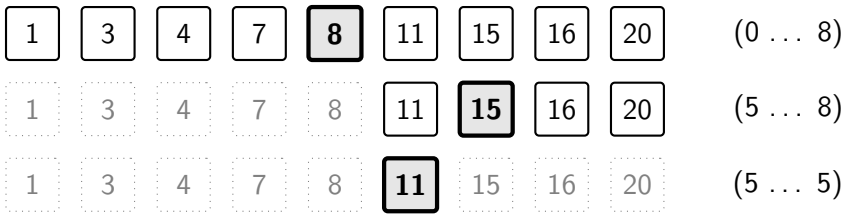
In dieser Aufgabe sollen Sie ein Verfahren programmieren, mit dem sich eine sortierte Liste effizient nach einem bestimmten Eintrag durchsuchen lässt. Geht man die Liste einfach (linear) Element für Element durch, nimmt das manchmal sehr viel Zeit in Anspruch. Nutzt man aber aus, dass die Liste sortiert ist, lässt sich die Suche ganz erheblich beschleunigen.

In dieser Aufgabe beschränken wir uns auf *aufsteigend* sortierte Listen mit `int`-Werten.

Der Algorithmus überprüft zunächst die Zahl in der Mitte der Liste.

- Ist sie die gesuchte Zahl, ist die Suche beendet.
- Ist sie größer als die gesuchte Zahl, muss die gesuchte Zahl — wenn sie überhaupt enthalten ist — in der linken Hälfte der Liste zu finden sein, die rechte Hälfte muss nicht mehr durchsucht werden.
- Ist sie kleiner als die gesuchte Zahl, wird die Suche entsprechend in der rechten Hälfte rekursiv fortgesetzt.

Beispiel: Suche nach der Zahl 11



8 ist in der Mitte der Liste und wird zuerst überprüft. Da $8 < 11$, wird in der rechten Hälfte der Liste weitergesucht. Da diese Teilliste vier Elemente hat, gibt es keine eindeutige Mitte, unser Algorithmus hat in diesem Fall die 15 ausgewählt. Diese ist nun größer als 11, die 11 muss also weiter links stehen, wo sie im nächsten Schritt auch tatsächlich gefunden wird.

Was ist ein geeigneter Abbruchfall, falls die Zahl nicht in der Liste enthalten ist?

Legen Sie eine Datei `search.py` an und implementieren Sie den Algorithmus in einer Funktion `contains`, die eine aufsteigend sortierte Liste von `int`s und einen `int`-Wert übergeben bekommt und `True` zurückgibt, wenn das gesuchte Element in der Liste enthalten ist, andernfalls `False`.

Anstatt tatsächlich für jeden Aufruf eine Kopie der entsprechenden Teil-Liste zu erzeugen, soll die Funktion neben der Liste und dem gesuchten Element zwei optionale Parameter `lo` und `hi` erhalten. `lo` ist der Index des ersten Elementes der zu durchsuchenden Teil-Liste, `hi` der Index des letzten. (Für eine vollständige Liste der Länge 5 wären die Werte also `lo`=0, `hi`=4.)

Wie könnte man mit diesen beiden Zahlen ausdrücken, dass eine Teil-Liste leer ist?

Falls vom Aufrufer keine Werte für `lo` und `hi` übergeben werden, soll die Funktion die gesamte Liste durchsuchen.

Testen Sie Ihre Funktion mit unterschiedlichen Listen und Elementen an unterschiedlichen Positionen. Geben Sie danach die Datei `search.py` ab.