ETH zürich
Departement Informatik

Programmieren und Problemlösen

Dr. Dennis Komm
https://lec.inf.ethz.ch/ppl/
17. Mai 2021

Übungsblatt 5
Lösungsvorschläge

Aufgabe 1

Schreiben Sie ein Programm, welches per Divide-and-Conquer rekursiv das Maximum der Liste findet und zurückgibt.

Hinweis: Divide-and-Conquer ist eine Technik, bei der ein Problem in kleinere Teilprobleme zerlegt wird, die danach leichter zu lösen sind. Es ist beispielsweise einfach das Maximum einer Liste der Länge 1 zu bestimmen.

```
1 def maxi(daten):
2     # Ihr Code hier
```

Lösung

Die Funktion teilt die gegebene Liste wieder in zwei Teillisten auf und arbeitet rekursiv mit diesen weiter.

```
1  def maxi(daten):
2      if len(daten) == 1:
3          return daten[0]
4      mid = len(daten) // 2
5      left = maxi(daten[:mid])
6      right = maxi(daten[mid:])
7      if left > right:
8          return left
9      else:
10         return right
11
12 print(maxi([2, 3, 6, 1, 7, 8, 10, 14, 60, 35]))
```

Aufgabe 2

In der folgenden Implementierung von Quicksort gibt es einige Fehler. Versuchen Sie, diese zu finden und zu korrigieren.

```python
def quicksort(daten):
    if len(daten) > 1:
        return daten
    pivot = daten[0]
    daten.pop(0)
    left_part = [0 for l in daten if l < pivot]
    right_part = [l for l in daten if l > pivot]
    left_sorted = quicksort(right_part)
    right_sorted = quicksort(left_part)
    return left_sorted + pivot + right_sorted

a = [0, 0, 0, 0, 0, 0, 0, 0, 0]
b = [4, 1, 5, 2, 6, 7, 8, 9, 0]
c = [4, 5, 3, 7, 4, 6, 7, 4, 9]
d = []

print(quicksort(d))
```

Lösung

Im Folgenden kommentieren wir die Korrekturen direkt im Code.

```python
def quicksort(daten):
    if len(daten) < 1:
        return daten
    pivot = daten[0]
    daten.pop(0)
    # We want to have l in this list and not 0
    left_part = [l for l in daten if l < pivot]
    # The identical elements also need to be checked
    right_part = [l for l in daten if l >= pivot]
    # Recursive call on the left side
    left_sorted = quicksort(left_part)
    # Recursive call on the right side
    right_sorted = quicksort(right_part)
    # The pivot needs to be a list as well -> [pivot]
    return left_sorted + [pivot] + right_sorted
```

Aufgabe 3

Vollziehen Sie die folgende Implementierung zur Berechnung der Fibonaccizahlen nach. In der Vorlesung haben Sie bereits zwei Varianten dieser Funktion gesehen, wobei die eine Mühe hatte, grosse Fibonaccizahlen zu berechnen. Erklären Sie, wieso die hier gezeigte Implementierung auch für `fibonacci(100)` gut funktioniert.

```python
def fibonacci(n):
    if n == 1:
        return (1, 0)
    fib1, fib2 = fibonacci(n-1)
```

```
    return fib1 + fib2, fib1

for i in range(1,20):
    a, b = fibonacci(i)
    print(a)
```

Lösung

Im Gegensatz zu der in der Vorlesung vorgestellten Version, gibt es in dieser Implementierung nur einen rekursiven Aufruf. Anstelle eines Dictionarys, in dem alle Zahlen gespeichert werden, wird hier immer die aktuelle Fibonaccizahl und die Fibonaccizahl davor übergeben. Diese beiden Zahlen reichen aus, um die nächste Zahl zu berechnen.

Aufgabe 4

Schreiben Sie eine Funktion, die eine Liste als Parameter übergeben bekommt und daraus ein Anagramm erstellt. Verwenden Sie dazu Rekursion.

```
def list_mirror(daten):
    # Ihr Code hier

print(list_mirror([1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

Ausgabe:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Lösung

Die Funktion kann folgendermassen erstellt werden.

```
def list_mirror(daten):
    if len(daten) == 0:
        return []
    return [daten[0]] + list_mirror(daten[1:]) + [daten[0]]

print(list_mirror([1,2,3,4,5,6,7,8,9]))
```

Aufgabe 5

In dieser Aufgabe sollen Sie ein Programm schreiben, das ein Dictionary wie ein Telefonbuch verwendet.

Hinweis: Stellen Sie sicher, dass ein Wert im Telefonbuch vorhanden ist, bevor Sie ihn löschen.

```
def add(telefonbuch):
    # Ihr Code hier

def delete(telefonbuch):
    # Ihr Code hier
```

```
6
7  def drucken(telefonbuch):
8      # Ihr Code hier
9
10 telefonbuch = dict()
11 while True:
12     next = input("Add, Delete, Print oder Stop? ")
13     if next == "Stop":
14         break
15     elif next == "Add":
16         telefonbuch = add(telefonbuch)
17     elif next == "Print":
18         telefonbuch = drucken(telefonbuch)
19     elif next == "Delete":
20         delete(telefonbuch)
```

Lösung

Es gibt verschiedene Möglichkeiten, diese Funktionen zu implementieren. Wichtig ist nur, dass man überprüft, ob ein Element, das man löschen möchte, auch wirklich im Dictionary enthalten ist. Dies wird hier in Zeile 9 gemacht. In Zeile 10 wird dann der Eintrag gelöscht.

In Zeile 4 wird ein Element in das Dictionary hinzugefügt. Dabei wird der Key in die eckigen Klammern geschrieben, und das Element mit dem Zuweisungsoperator übergeben.

In Zeile 17 sieht man, wie man alle Keys des Dictionary erhalten kann. Um das entsprechende Element zu erhalten, muss man mittels der eckigen Klammern darauf zugreifen (Zeile 18).

```
1  def add(telefonbuch):
2      name = input("Name: ")
3      nummer = int(input("Nummer: "))
4      telefonbuch[name] = nummer
5      return telefonbuch
6
7  def delete(telefonbuch):
8      name = input("Wer soll geloescht werden? ")
9      if name in telefonbuch.keys():
10         del telefonbuch[name]
11         print("Die Person mit namen", name, "wurde geloescht")
12     else:
13         print("Fehler, diese Person existiert nicht")
14     return telefonbuch
15
16 def drucken(telefonbuch):
17     for k in telefonbuch.keys():
18         print("Name:", k, ", Nummer:", telefonbuch[k])
19
20 telefonbuch = {}
21 while True:
22     next = input("Add, Delete, Print oder Stop? ")
23     if next == "Stop":
```

```
24          break
25      elif next == "Add":
26          telefonbuch = add(telefonbuch)
27      elif next == "Print":
28          drucken(telefonbuch)
29      elif next == "Delete":
30          telefonbuch = delete(telefonbuch)
```