TECHNISCHE UNIVERSITÄT DRESDEN

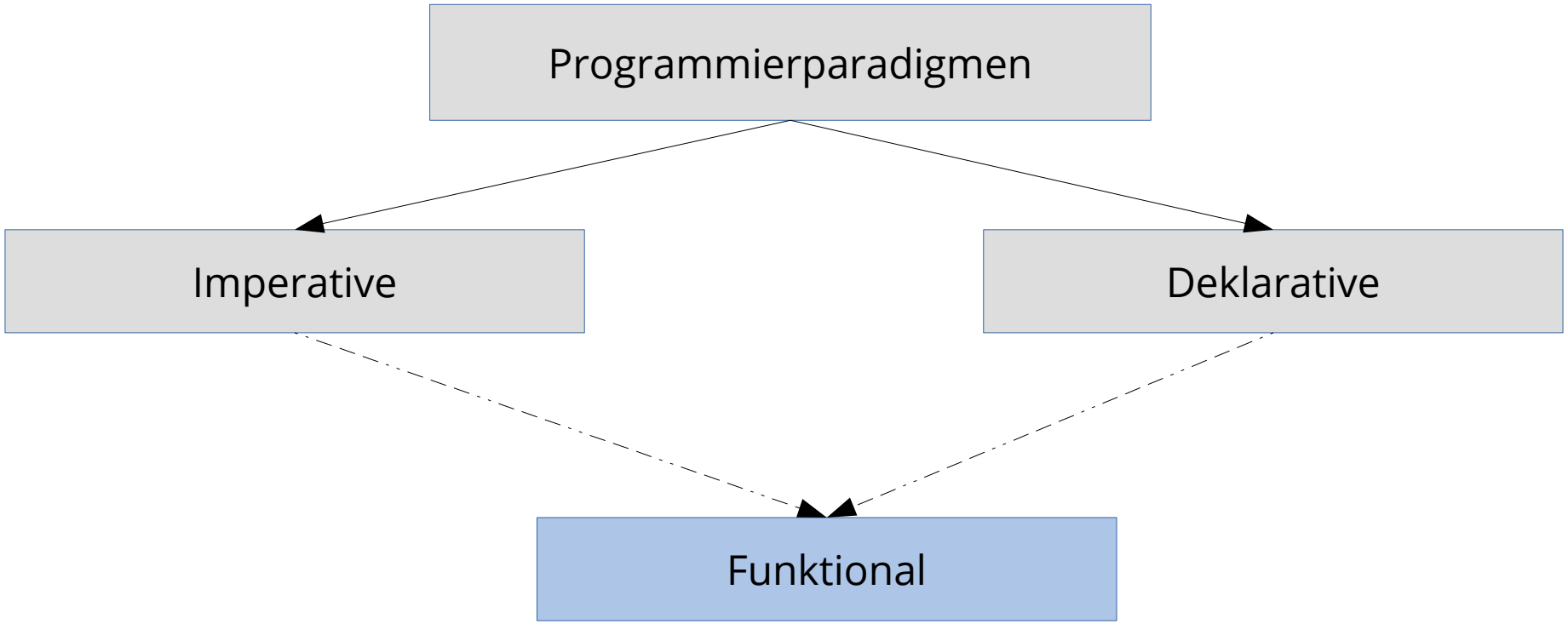
DRESDEN concept

Professur für Didaktik der Informatik
Dr. Thiemo Leonhardt

Programmierparadigmen
Rekursion

Professur für Didaktik der Informatik

Klassifikation von Programmiersprachen



Einfache Rekursion

Rekursion

- Technik aus der funktionalen Programmierung
  - Deklarative Beschreibung von Problemlösungen
  - Vereinfacht die Beschreibung von Algorithmen auf Datenstrukturen
  - Weniger speichereffizient als iterative Lösungen
- Benutzte Technik
  - Selbstaufruf einer Funktion im eigenen Funktionsrumpf

Einfache Rekursion

```
def func():
    print('Hallo')
    func()

func()
```

Wie sieht dies im Debugger aus?

Zusammenhang Schleifen (Iteration) und Rekursion

```
i = 0
while i < 5:
    print(i)
    i = i + 1
```

Zusammenhang Schleifen (Iteration) und Rekursion

```
def func(i):
    print(i)
    func(i+1)
```

```
def func(i):
    if i < 5:
        print(i)
        func(i+1)
```

Z nd Rekursion

Was bedeutet Rekursionsbaum?

```
def func(i):
    if i < 5:
        print(i)
        func(i+1)
```

Live Coding – Einfache Rekursion

Allgemeine Rekursion

Überblick

- Rekursion
  - Funktionsaufrufe in Funktionen
  - Basisfall
  - Rekursiver Fall
  - Rekursionsbaum

Ein erstes Programm

```
murmeln = [10, 13, 39, 14, 41, 9, 3]

print('Die Anzahl der Murmeln ist', sum(murmeln))
```

```
>>> %Run v1.py
 Die Anzahl der Murmeln ist 129
>>>
```

Ein erstes Programm

Eine Liste mit der Anzahl an Murmeln in den Taschen mehrerer Personen.

```
murmeln = [10, 13, 39, 14, 41, 9, 3]

print('Die Anzahl der Murmeln ist', sum(murmeln))
```

Benutzung von Python Build-In-Function sum, um die Murmeln zusammen zu zählen.

```
>>> %Run v1.py
 Die Anzahl der Murmeln ist 129
>>>
```

Iterative Lösung mit deklarativer Schleife

- Aufgabe
  - Implementieren Sie die Funktion *sum* in Python
  - Benutzen Sie die for…in-Schleife
  - Testen Sie ihre Funktion mit dem Murmelbeispiel


```
murmeln = [10, 13, 39, 14, 41, 9, 3]

print('Die Anzahl der Murmeln ist', sum(murmeln))
```

Mögliche Lösung

```
murmeln = [10, 13, 39, 14, 41, 9, 3]

def sum(list):
    summe = 0
    for i in list:
        summe = summe + i
    return summe

print('Die Anzahl der Murmeln ist', sum(murmeln))
```

Welche Funktion wird hier ausgeführt?

Ein neuer Lösungsweg

Nehmen wir an, die Python-Erfinder haben entschieden, die Iteration aus der Sprache zu entfernen (for und while).

Trotzdem müssen wir die Liste mit Zahlen berechnen. Geht das überhaupt ohne Iteration?

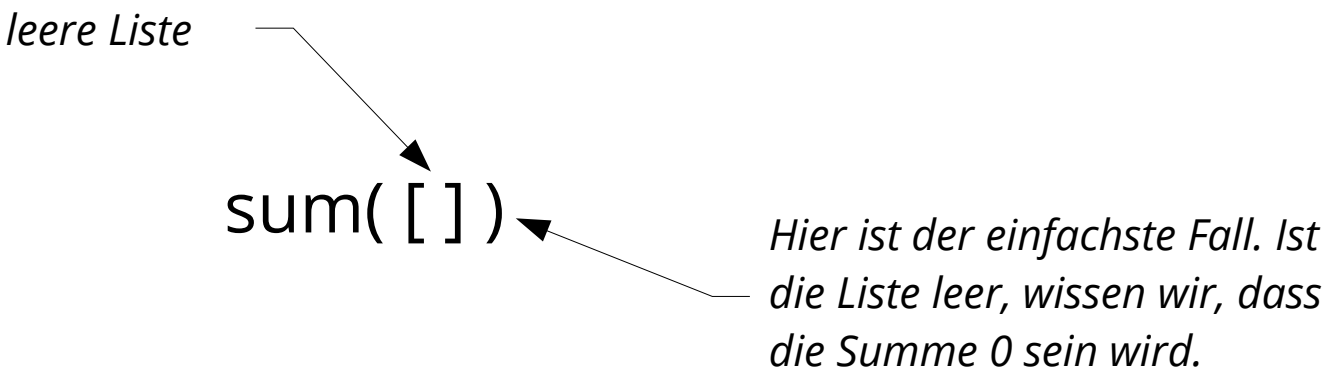
?

Ja! Durch Rekursion.

Nein, die eingebaute sum-Funktion dürfen Sie auch nicht benutzen.

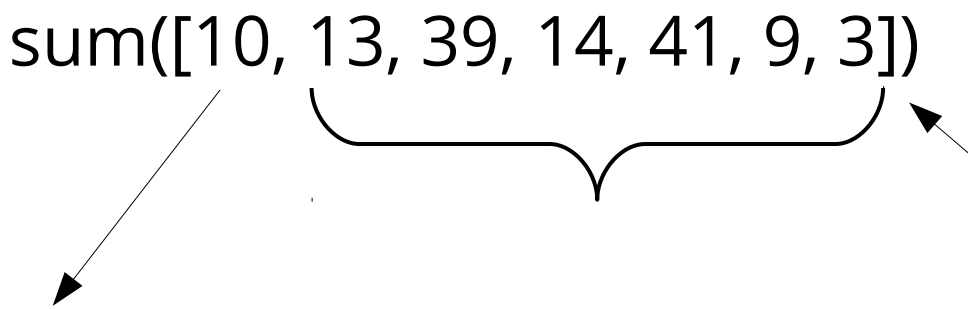
Zwei Fälle – Basisfall und Rekursiver Fall

Basisfall: Die wenigsten Probleme bereitet der einfachste vorstellbare Fall. Was ist der einfachste Fall einer Liste, deren Werte Sie addieren sollen?



Zwei Fälle – Basisfall und Rekursiver Fall

Rekursiver Fall: Hierbei lösen wir eine kleinere Version des gleichen Problems.



Wie können wir das Problem vereinfachen?
Wie wäre es mit der Summe der Liste, die um ein Element kürzer ist?

Hier haben wir unser Problem reduziert: Um die Summe der Liste zu berechnen, addieren wir 10 zu einer etwas kleineren Liste.

In unserem Beispiel gehen wir nach folgender Anweisung vor: Nimm das erste Listenelement und addiere es es zur Summe der verbleibenden Liste hinzu...

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

Basisfall: Überprüfen, ob die Liste leer ist, falls ja soll 0 zurückgegeben werden.

Aufgabe:

- Schreiben Sie eine Funktion *sum_rek*, die eine Liste als Parameter hat.
- Wenn die Liste leer ist soll die Funktion 0 zurückliefern.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

Basisfall: Überprüfen, ob die Liste leer ist, falls ja soll 0 zurückgegeben werden.

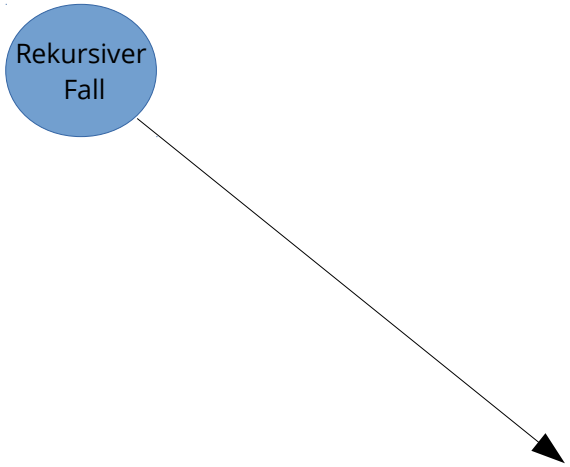
Lösung:

```
1  def sum_rek(liste):
2      if len(liste) == 0:
3          return 0
```

Hier wird überprüft, ob die Liste leer ist.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

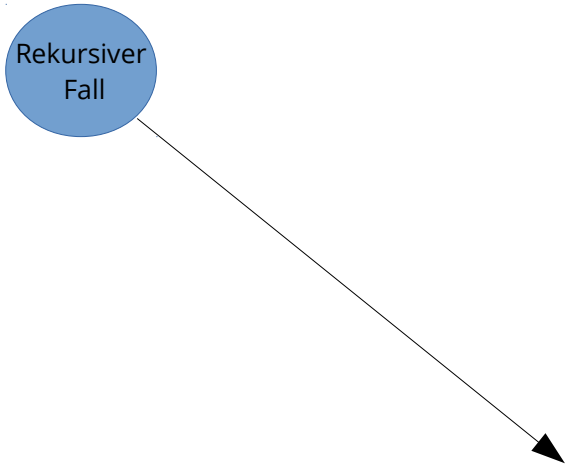



```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7
8
9
10
```

Eine Variable für das erste Listenelement.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

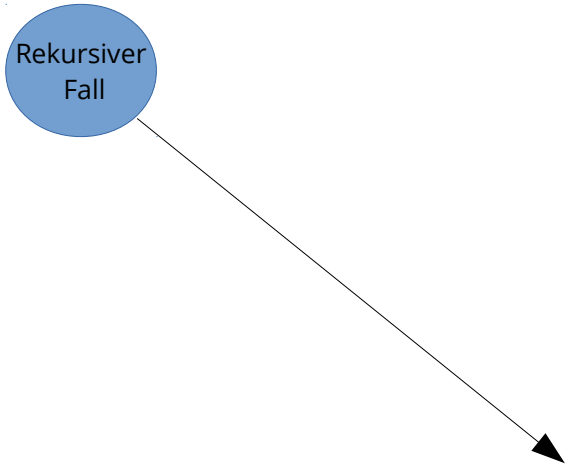



```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first =
8
9
10
```

Eine Variable für das erste Listenelement.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

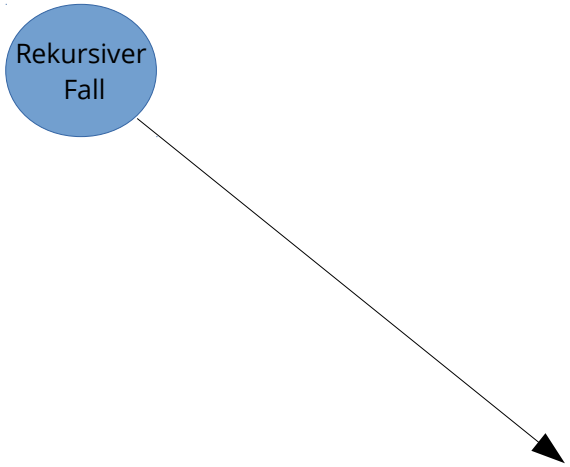



```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8
9
10
```

Eine Variable für das erste Listenelement.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

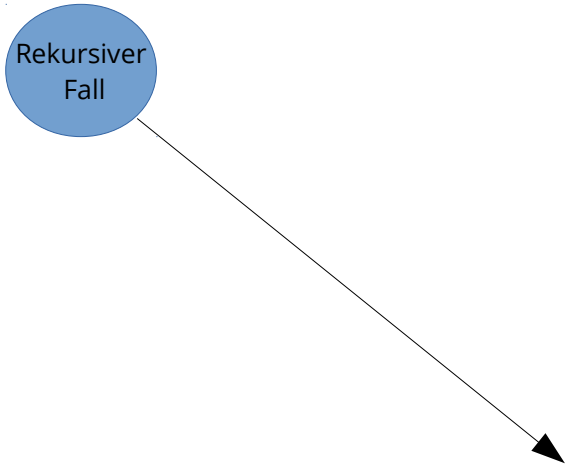



```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8
9
10
```

Entfernen des ersten Elements aus der Liste.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

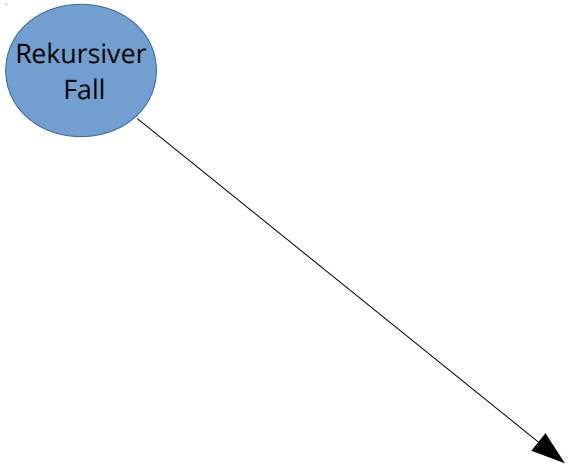



```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9
10
```

Entfernen des ersten Elements aus der Liste.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

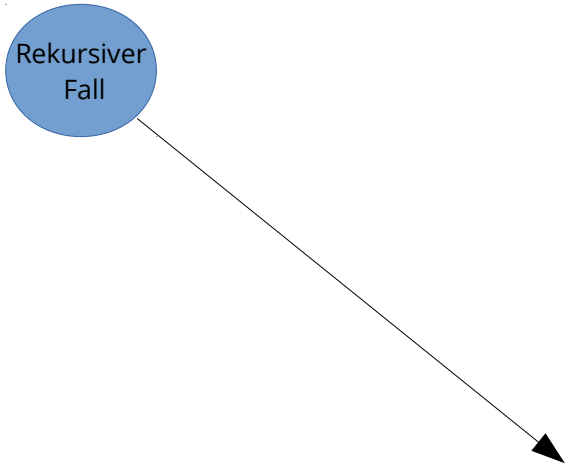



```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9
10
```

Jetzt muss das erste Element zur Summe der verbleibenden Elemente hinzuaddiert werden.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe =
10
```

Jetzt muss das erste Element zur Summe der verbleibenden Elemente hinzuaddiert werden.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

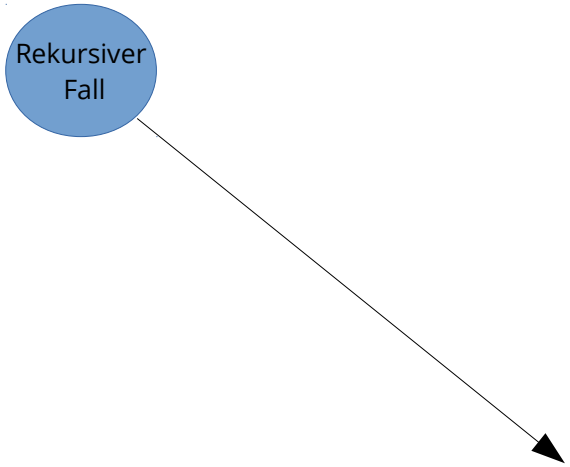
Rekursiver Fall

```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first +
10
```

Jetzt muss das erste Element zur Summe der verbleibenden Elemente hinzuaddiert werden.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

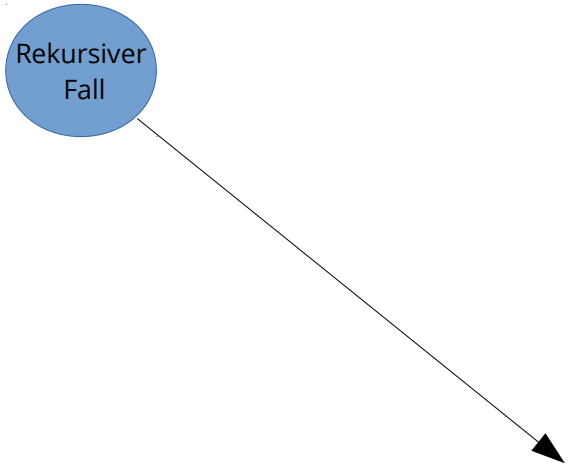



```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + Summe der verbleibenden Elemente
10
```

Jetzt muss das erste Element zur Summe der verbleibenden Elemente hinzuaddiert werden.

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.



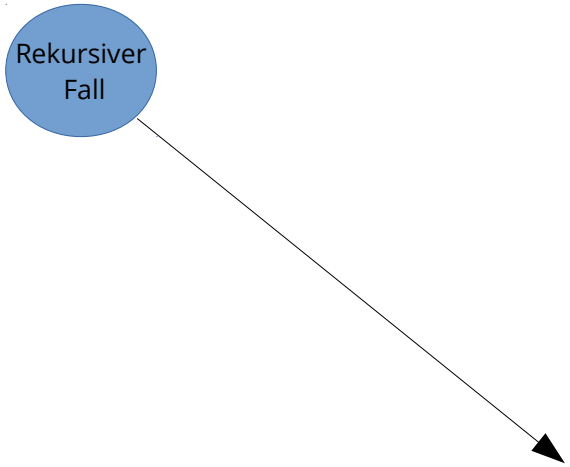

```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + Summe der verbleibenden Elemente
10
```

Die Summe ist das erste Element plus die Summe der verbleibenden Liste

Aber wie programmieren wir das?

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

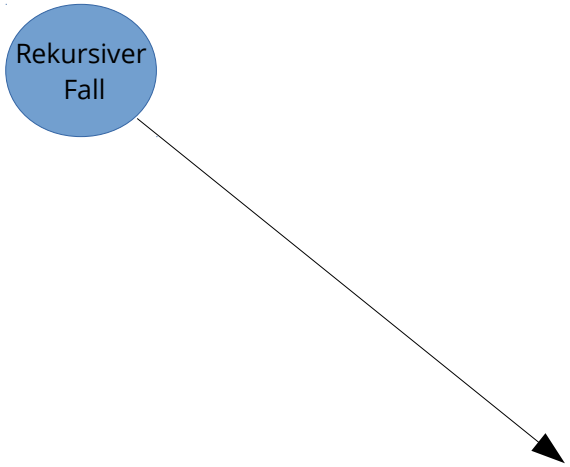



```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + Summe der verbleibenden Elemente
10
```

Ist das nicht genau das was wir gerade programmieren?

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

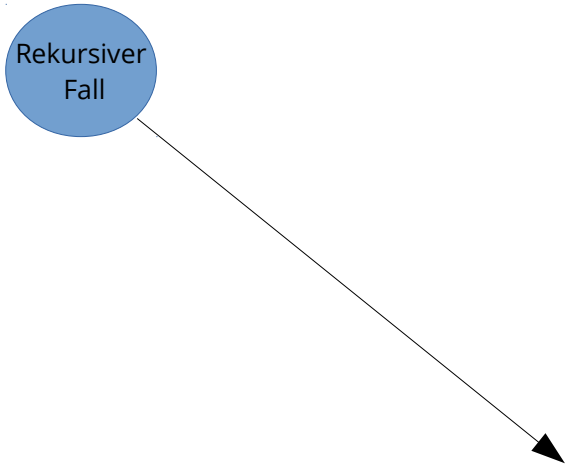



```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10
```

Ist das nicht genau das was wir gerade programmieren?

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.

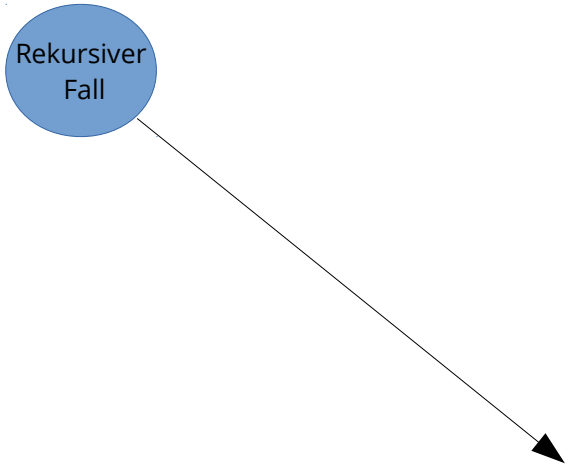



```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10
```

Irgendetwas fehlt noch?

Keine Angst vor dem Code

Basisfall und rekursiven Fall in Pythoncode umsetzen.




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Die Summe muss noch
zurückgegeben werden.

Keine Angst vor dem Code

Aufgabe: Probieren Sie es aus!

- Bringen Sie die Codestücke in die richtige Reihenfolge
- Ersetzen Sie den Funktionsaufruf an der richtigen Stelle

```
murmeln = [10, 13, 39, 14, 41, 9, 3]

print('Die Anzahl der Murmeln ist', sum(murmeln))
```

```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

Rekursionsbaum als Visualisierung

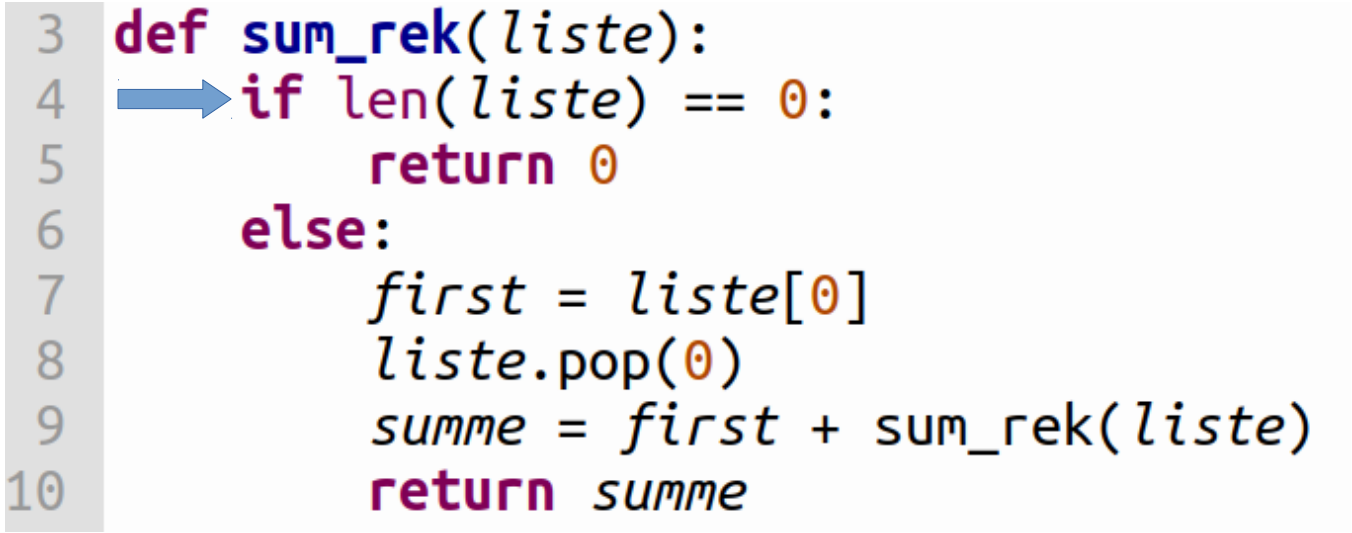
sum_rek([10, 13, 39, 14])

```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

Rekursionsbaum als Visualisierung

sum_rek([10, 13, 39, 14])

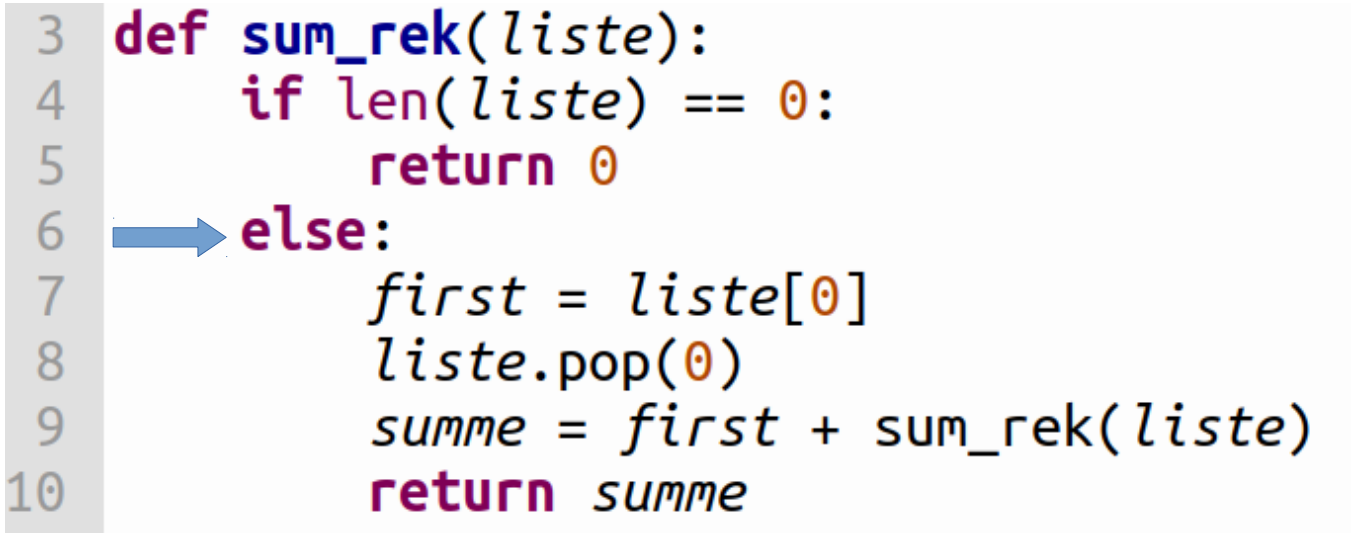


```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

Rekursionsbaum als Visualisierung

sum_rek([10, 13, 39, 14])

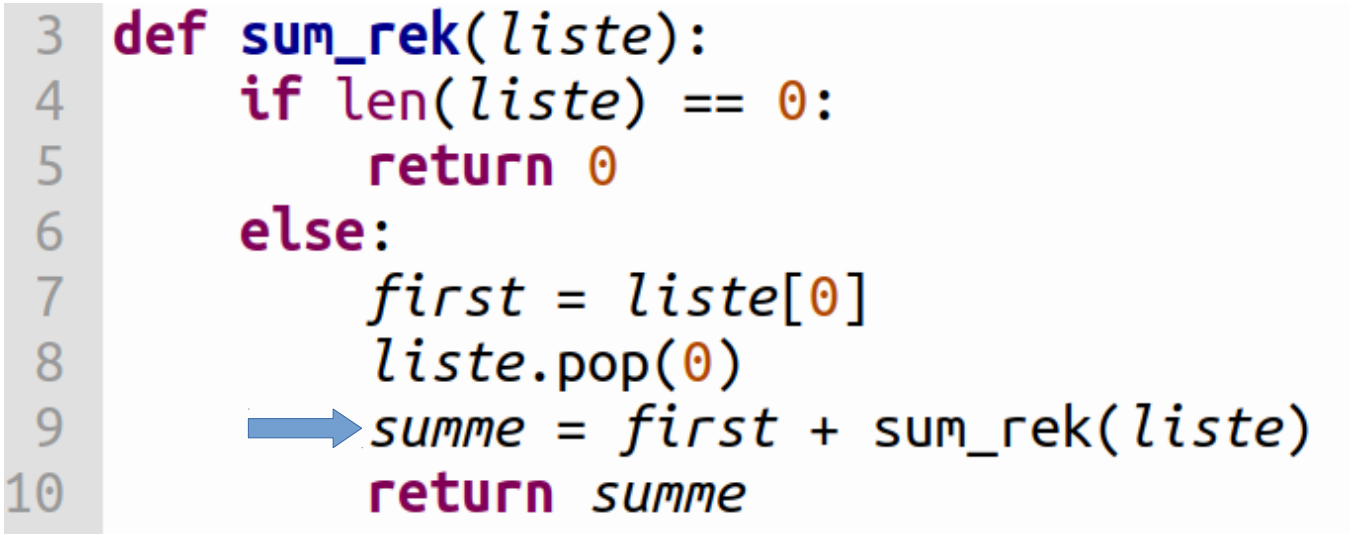


```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

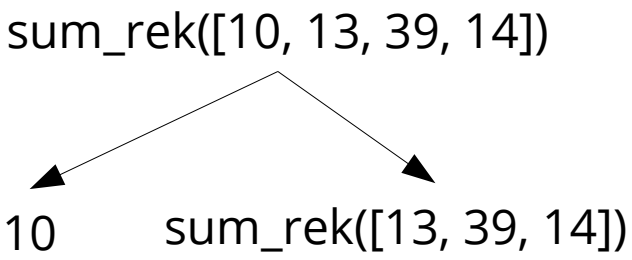
Rekursionsbaum als Visualisierung

sum_rek([10, 13, 39, 14])



Warum funktioniert das?

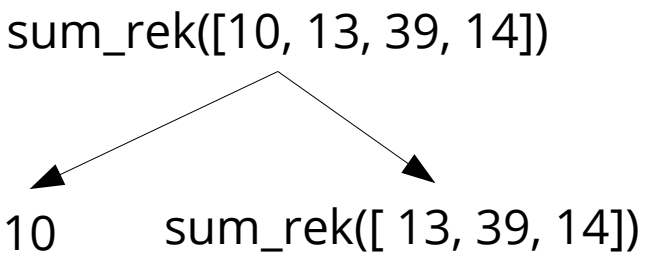
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

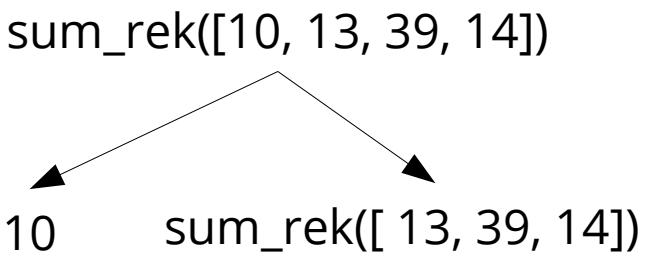
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

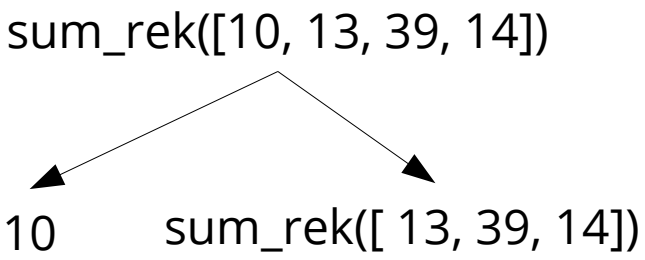
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

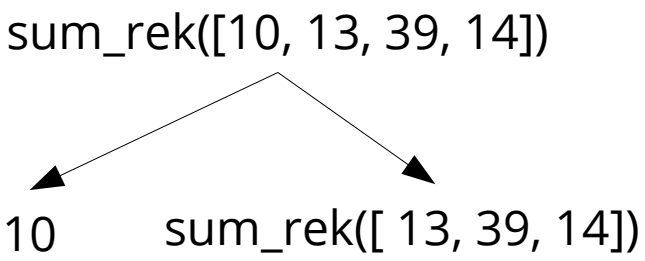
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

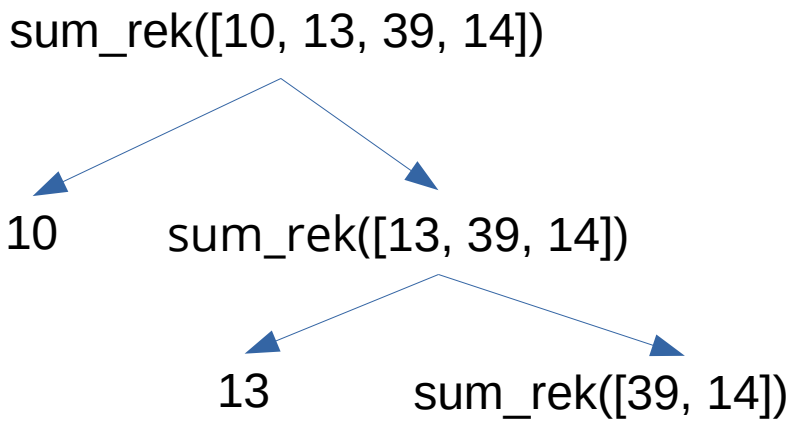
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

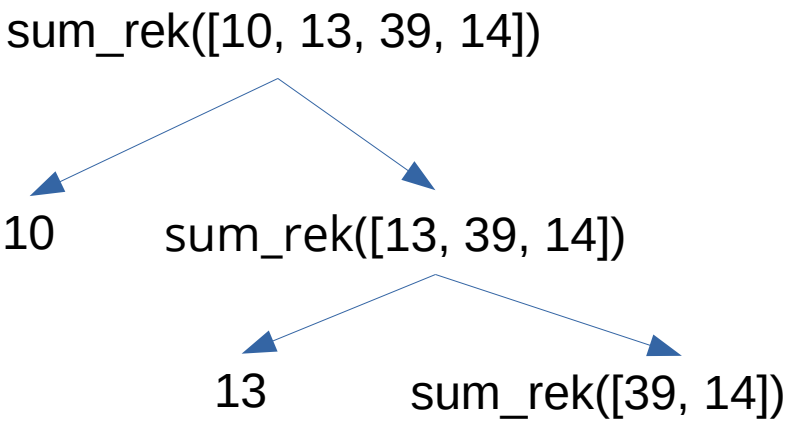
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

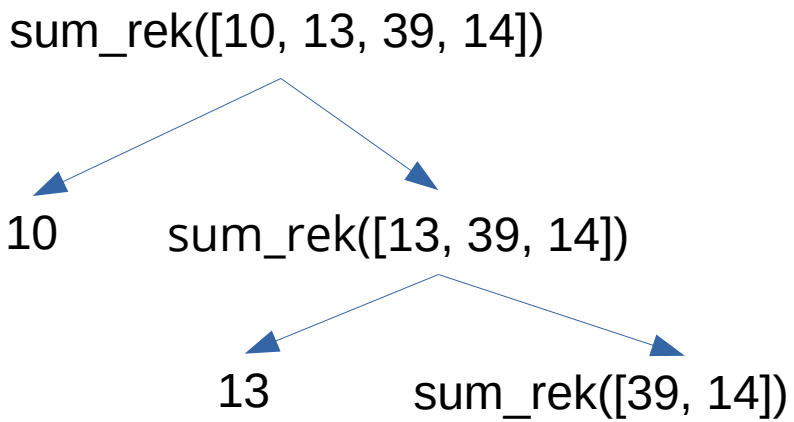
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

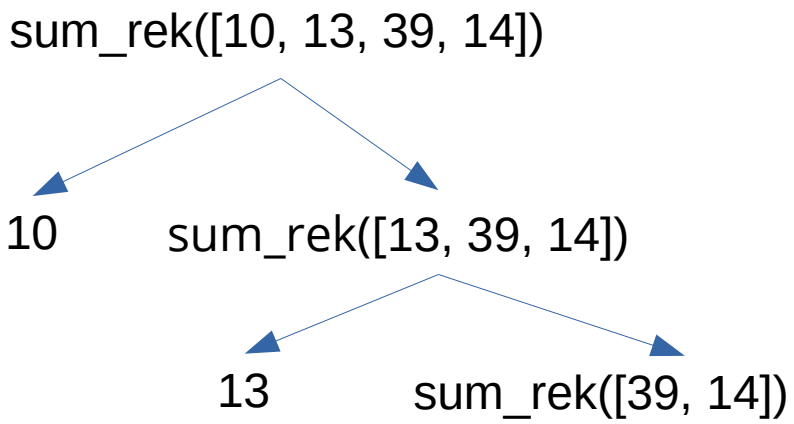
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

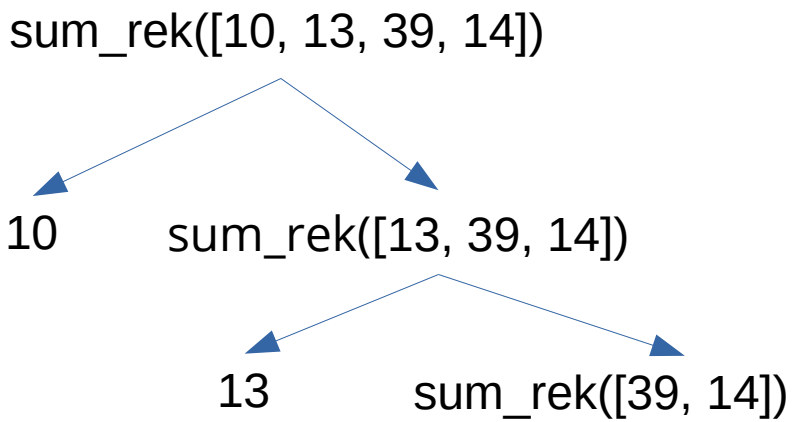
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

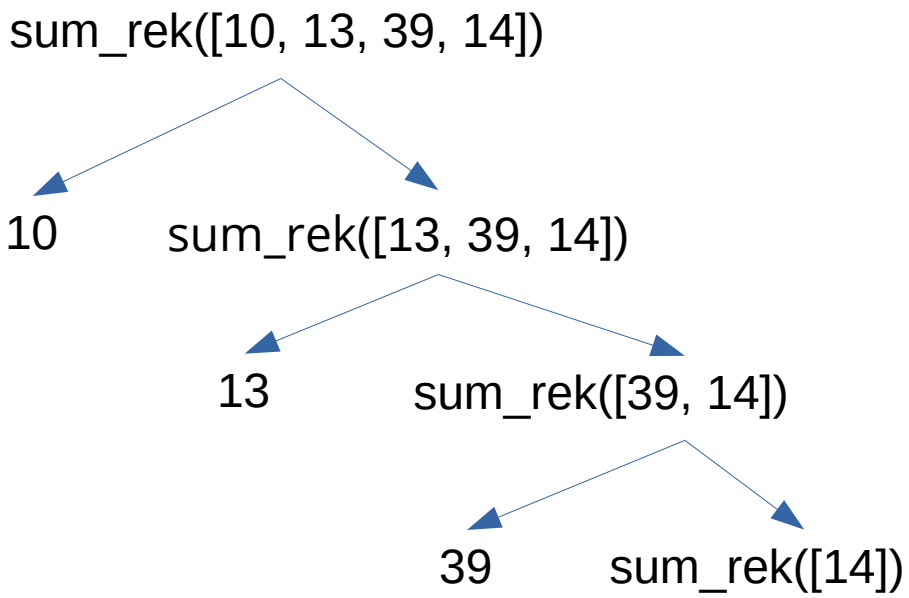
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

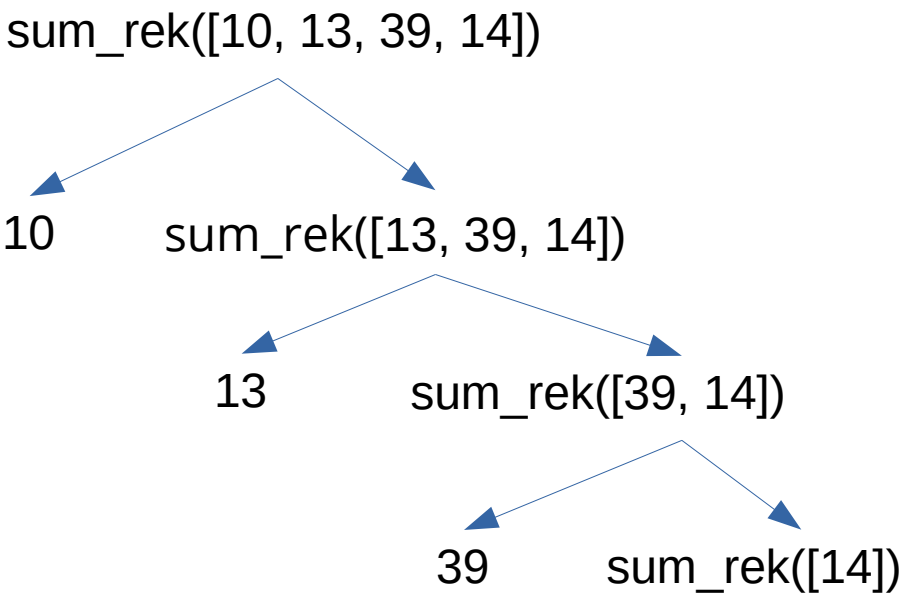
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

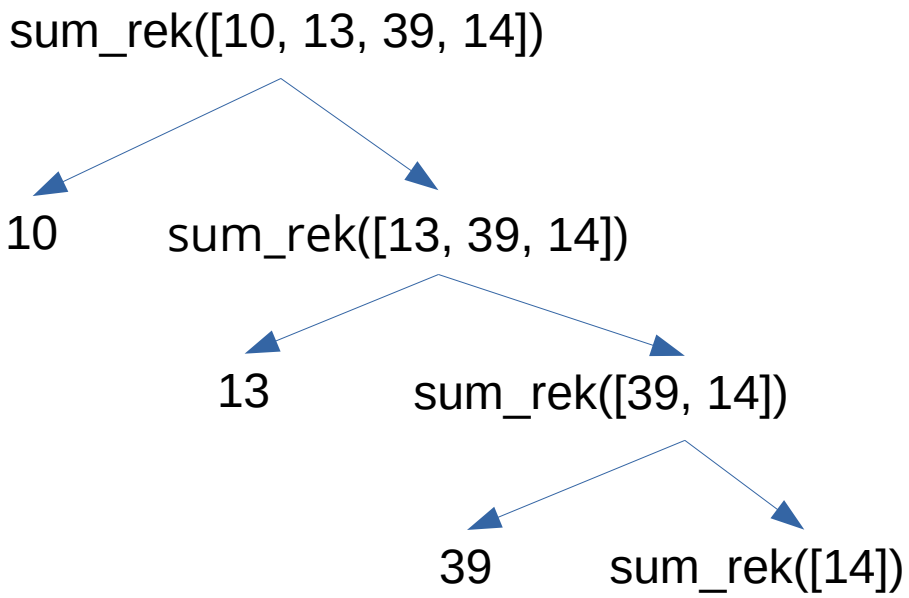
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

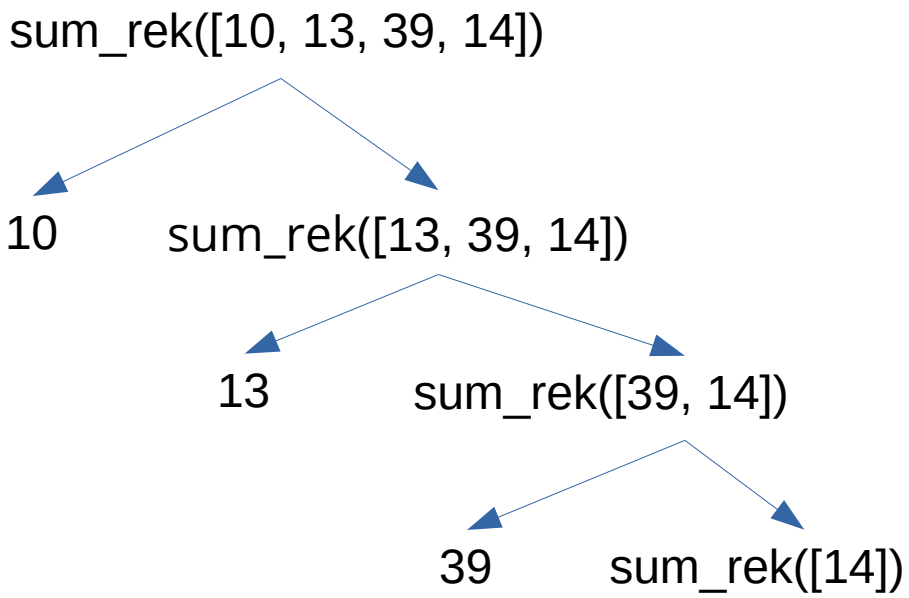
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

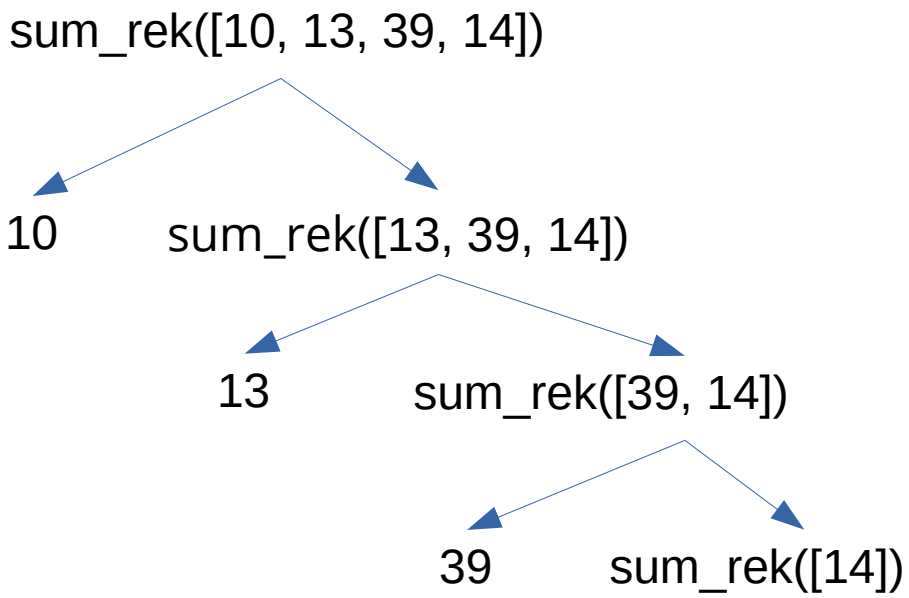
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

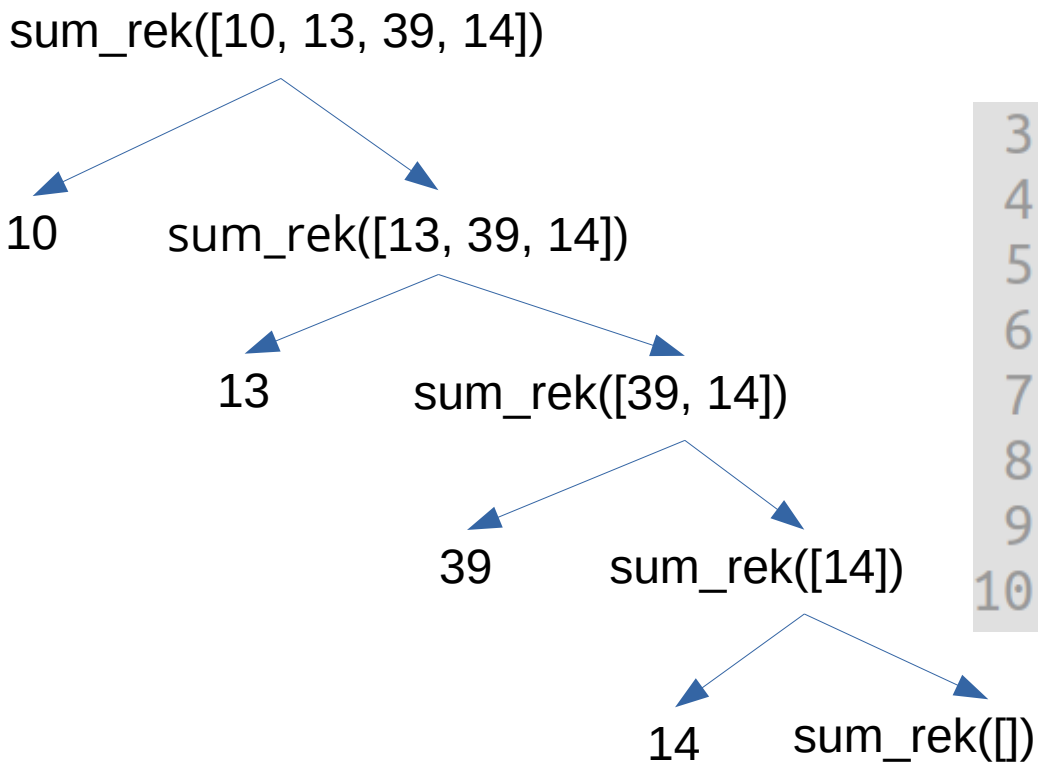
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

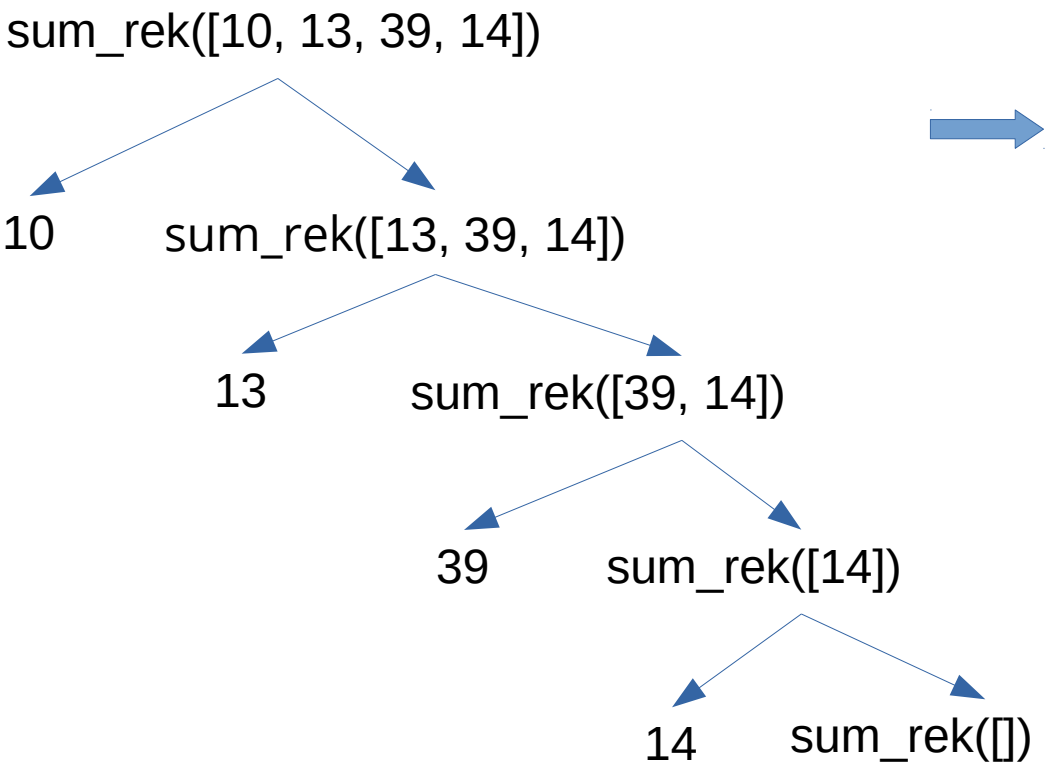
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

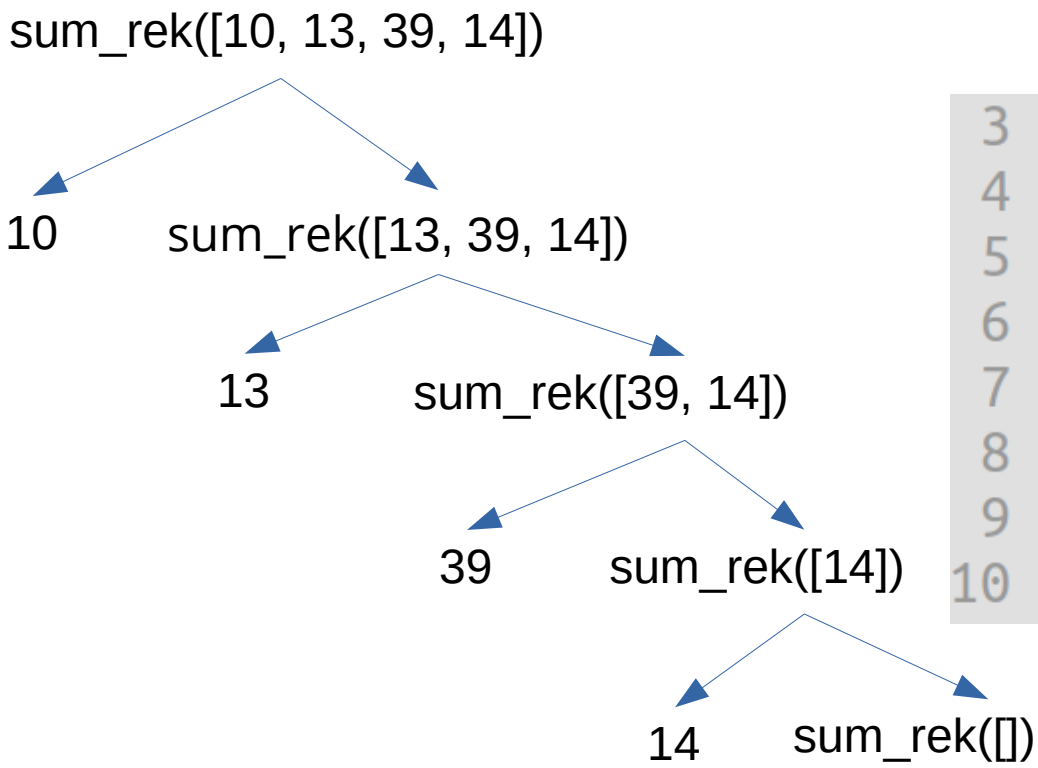
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

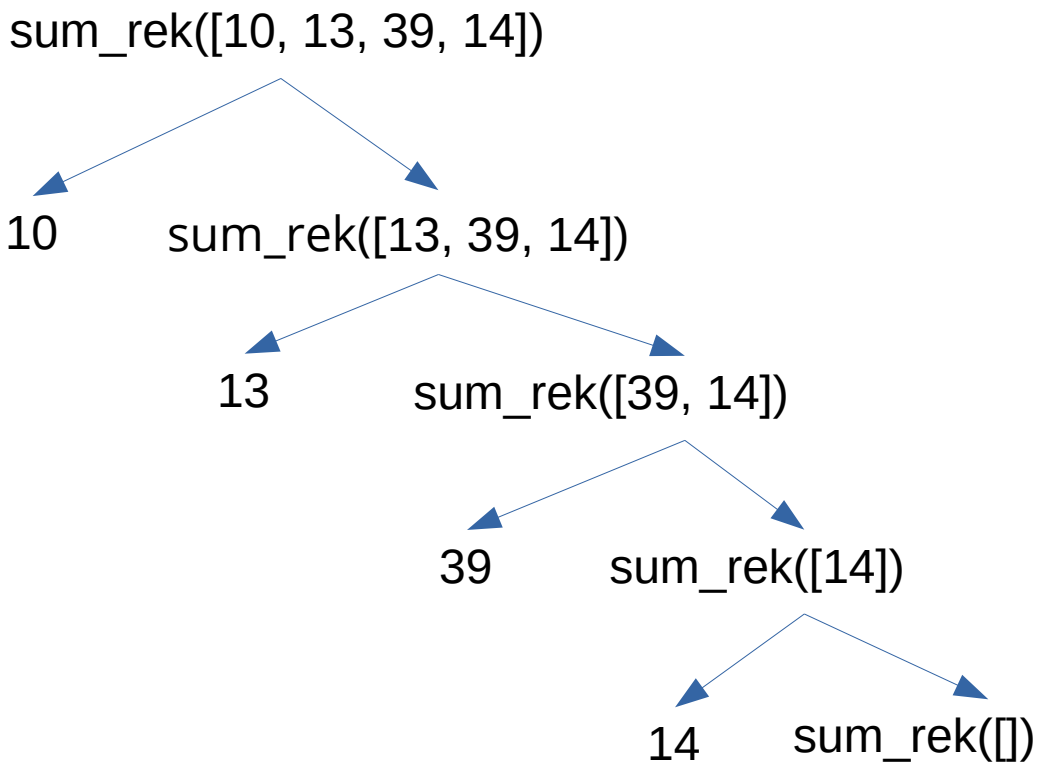
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

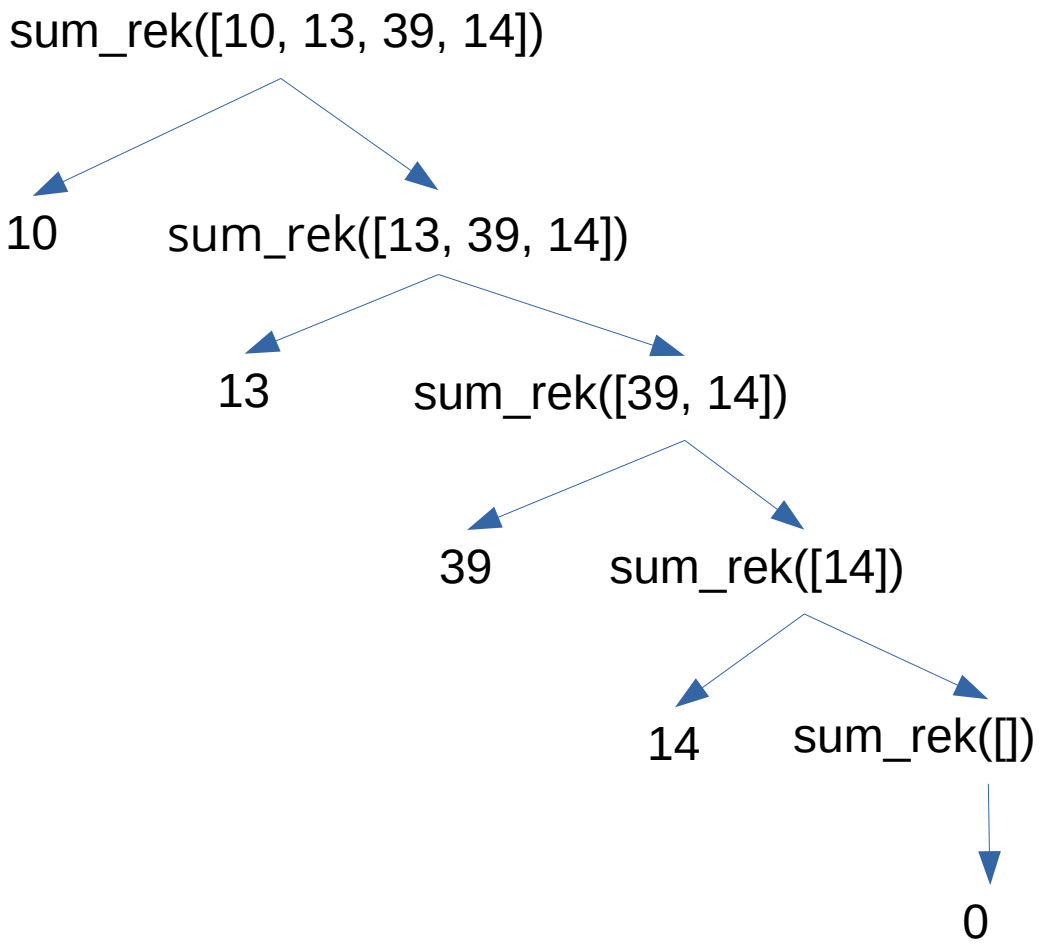
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

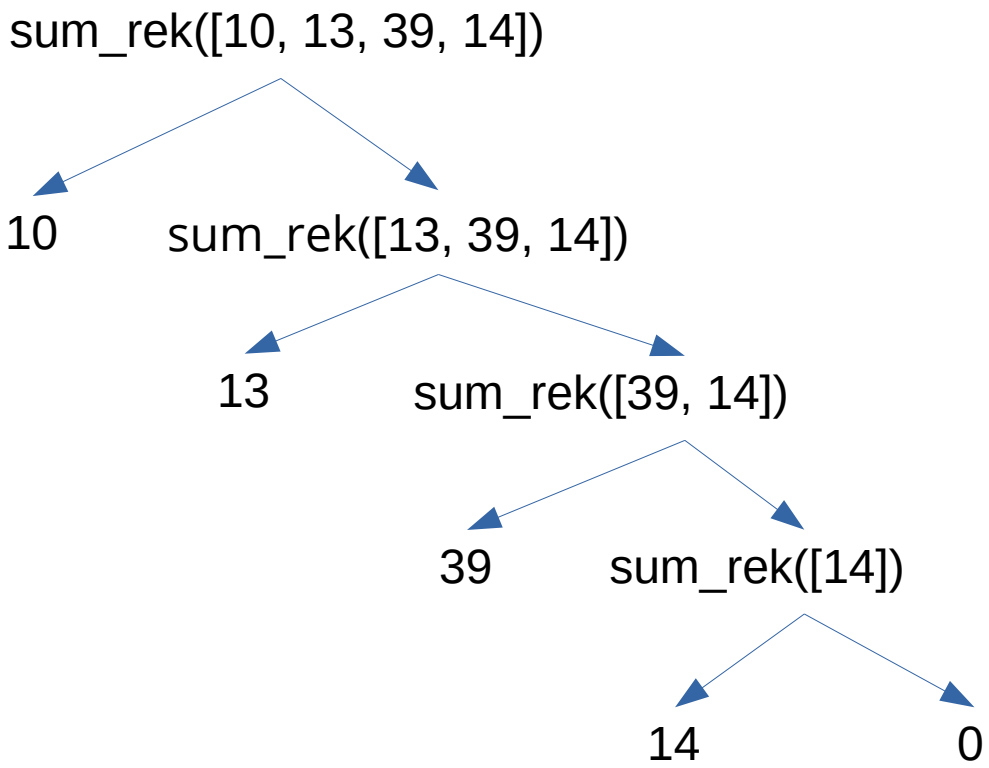
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

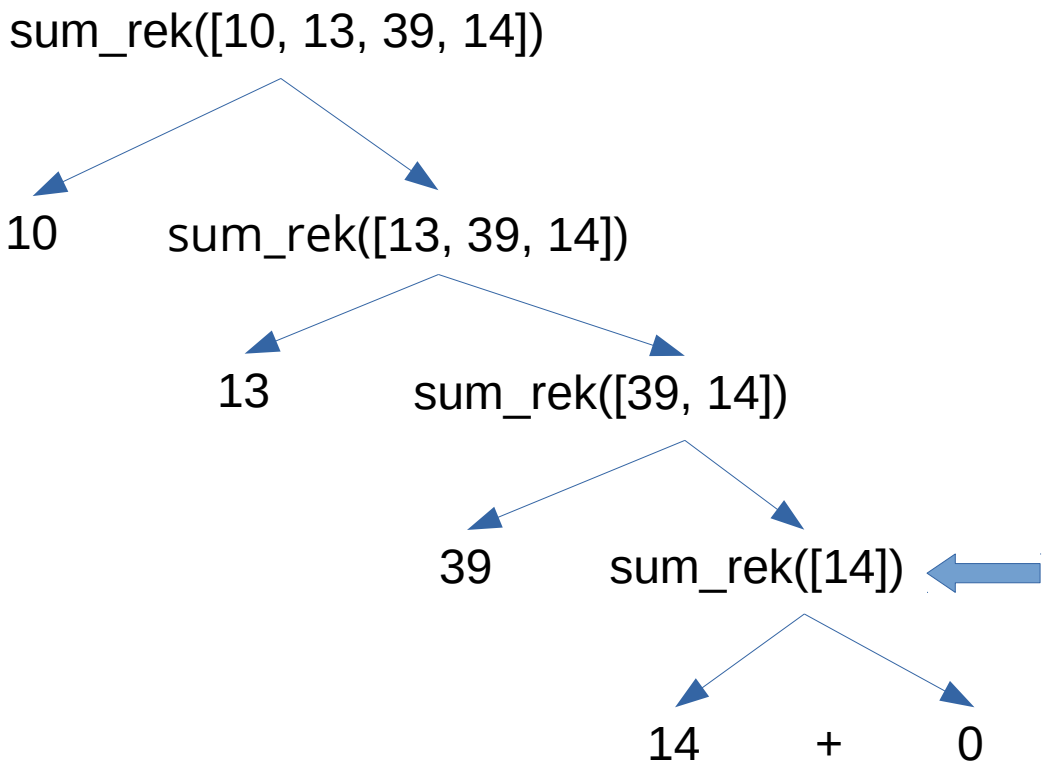
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

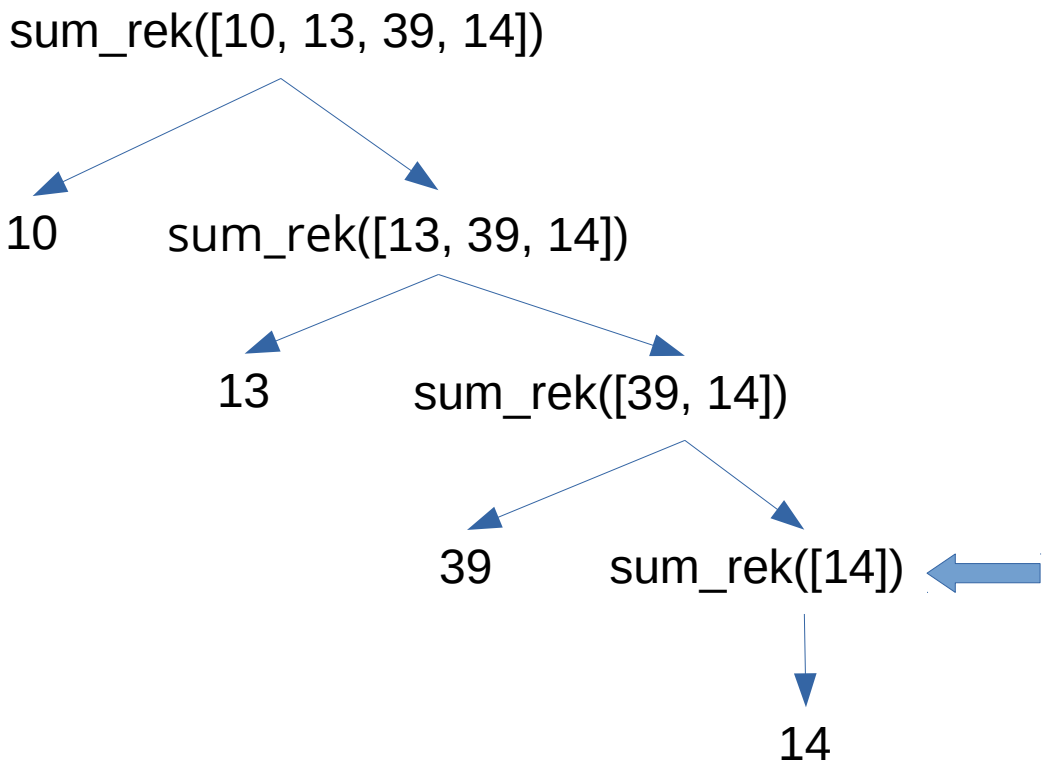
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

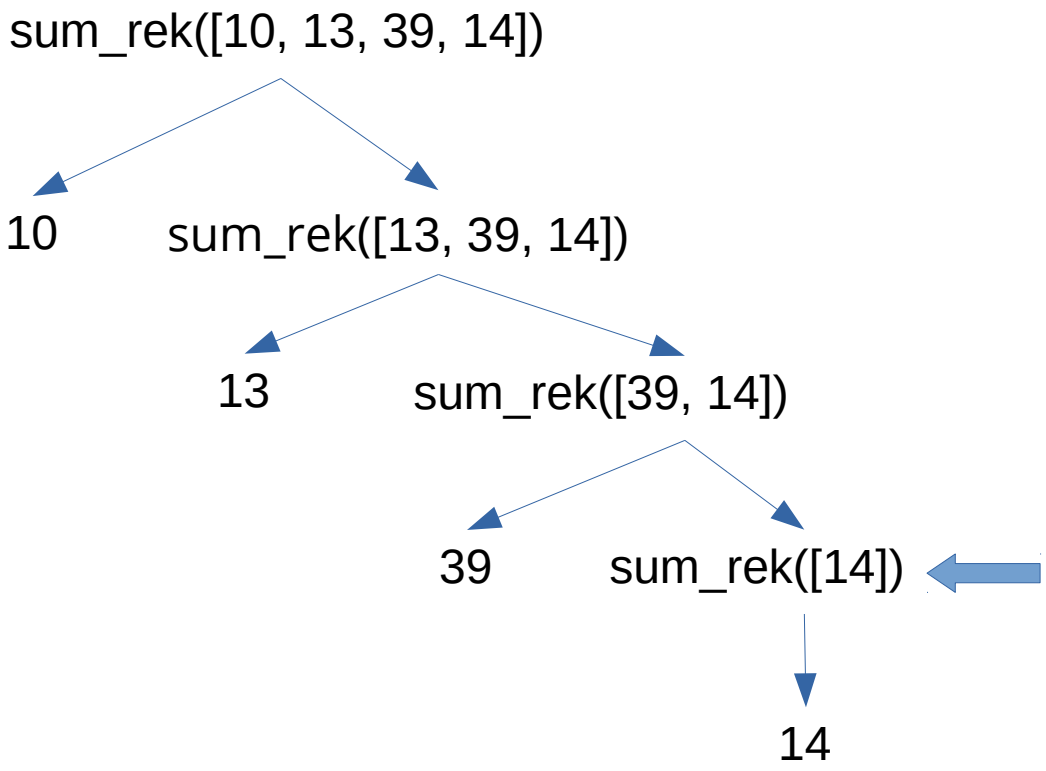
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

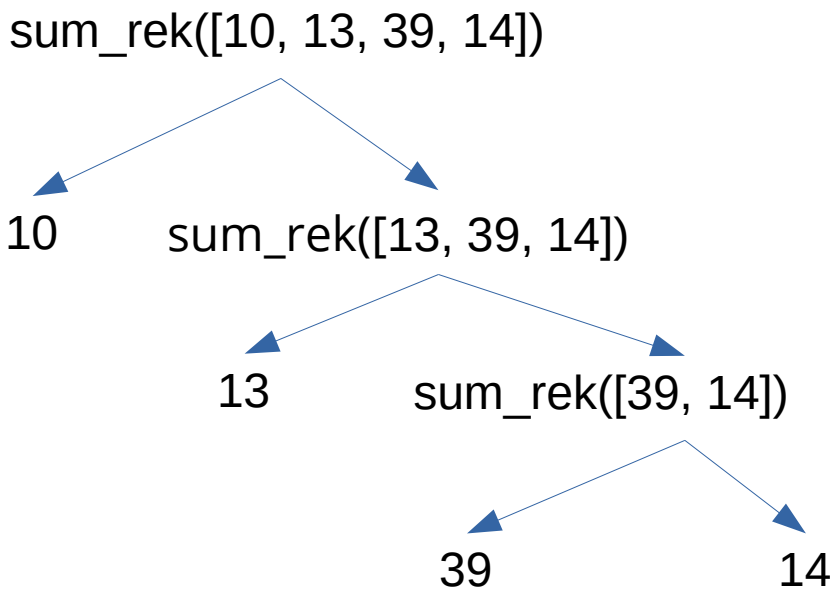
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

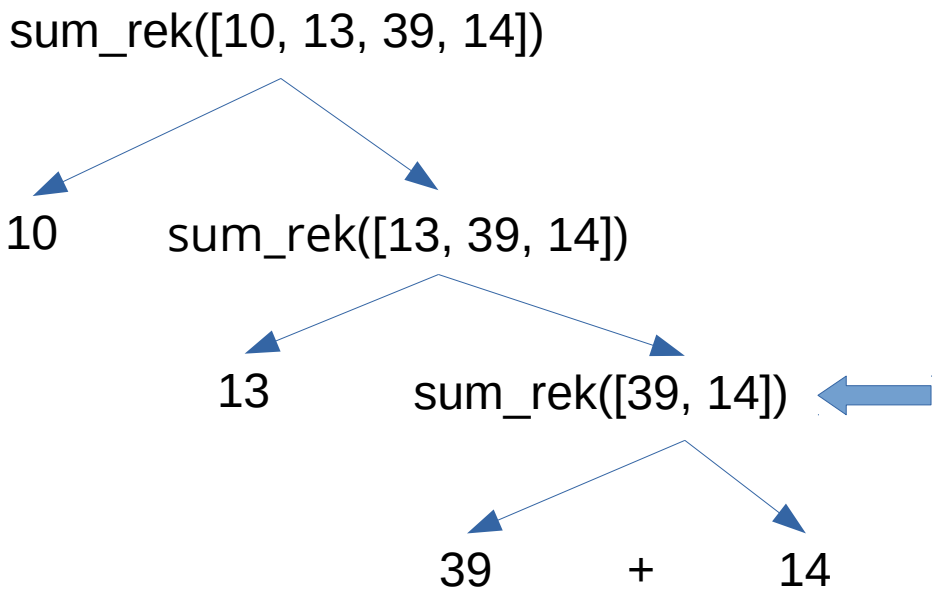
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

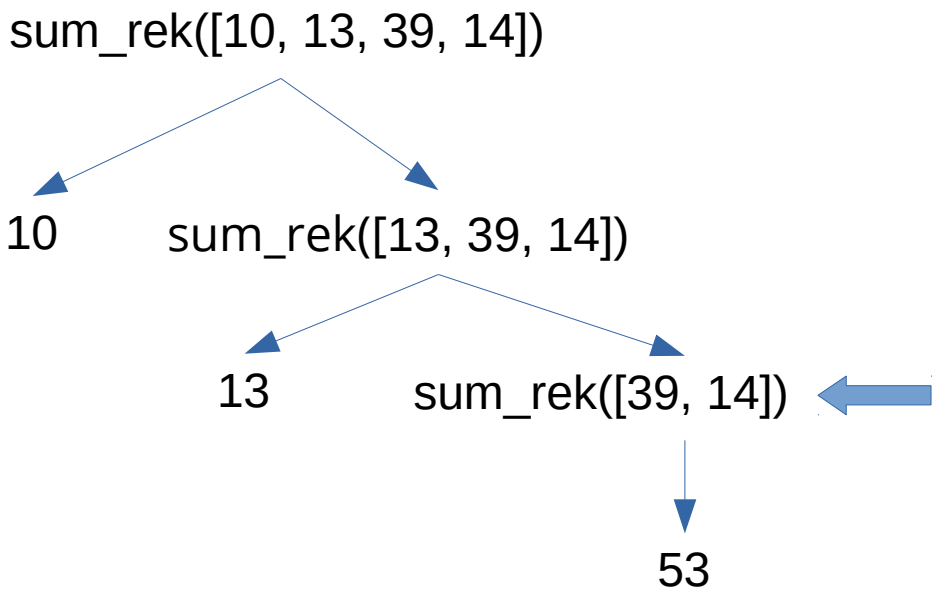
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

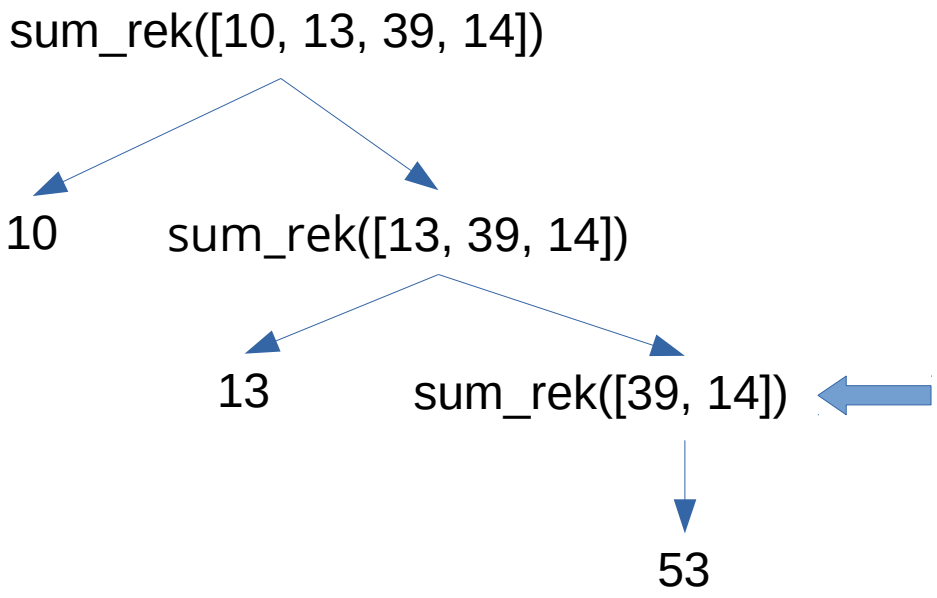
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

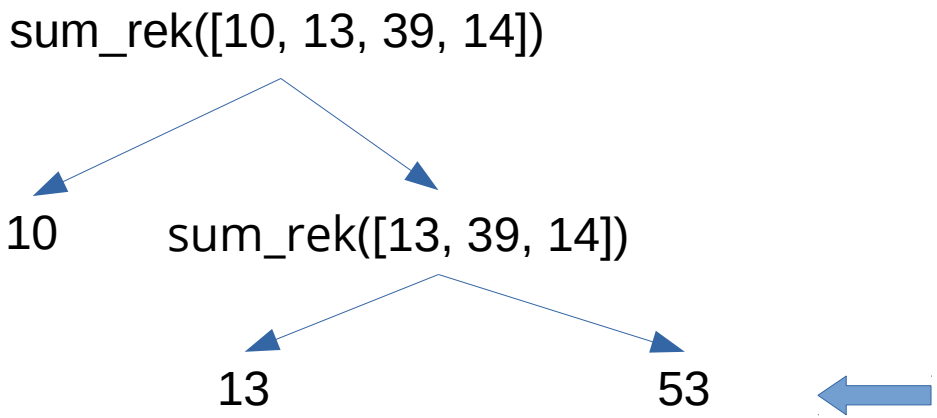
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

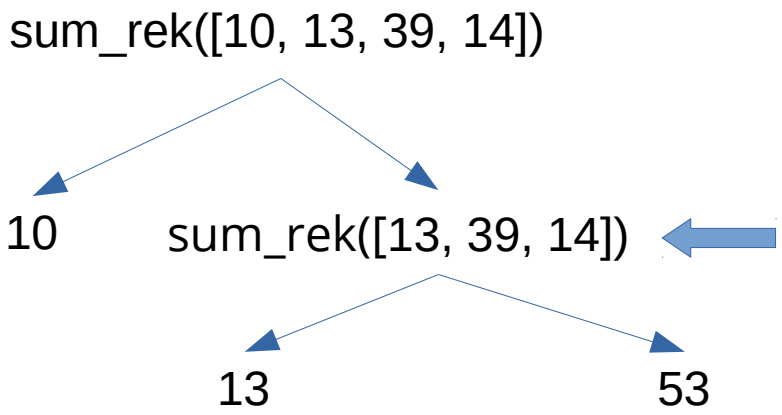
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

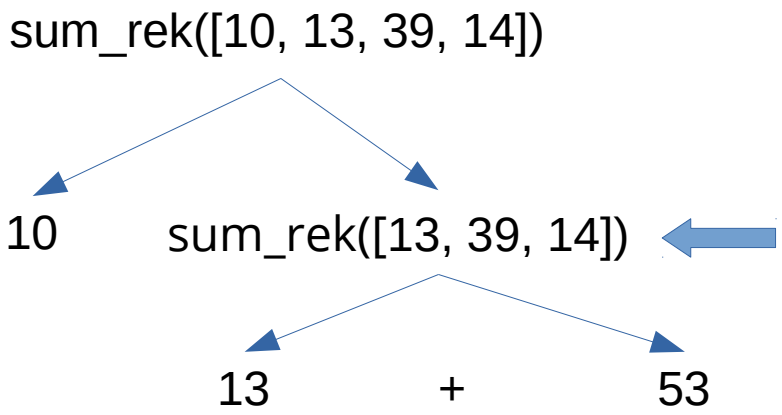
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

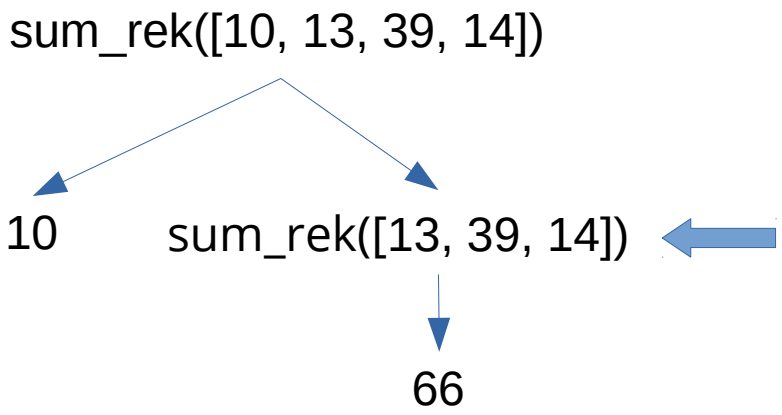
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

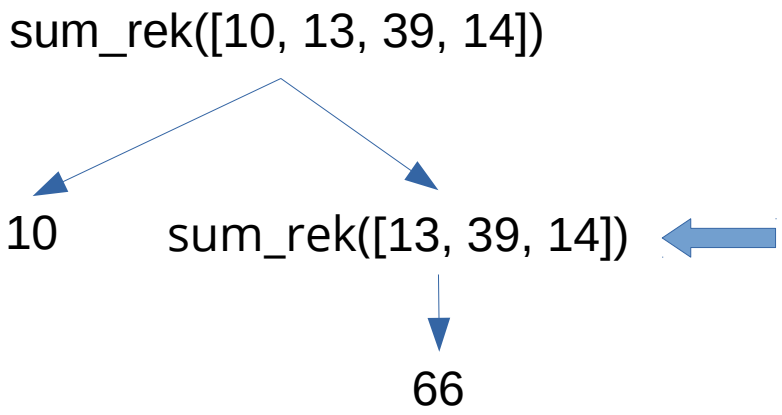
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

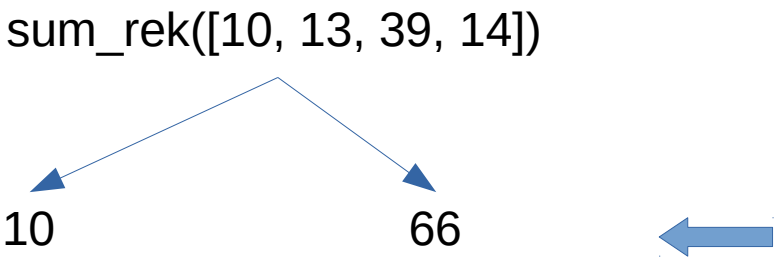
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

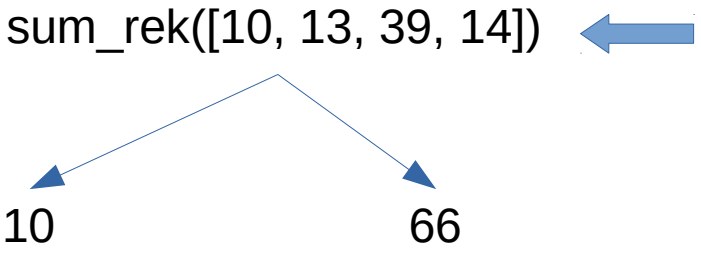
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

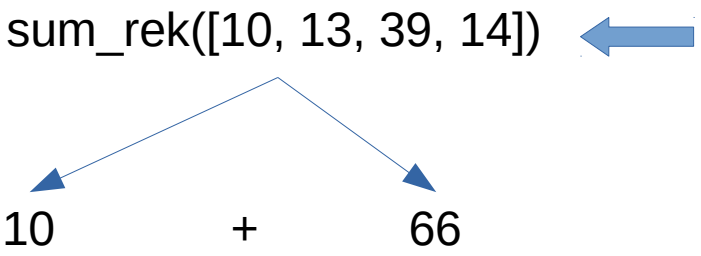
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

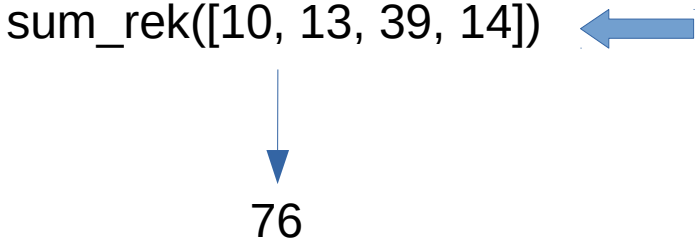
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

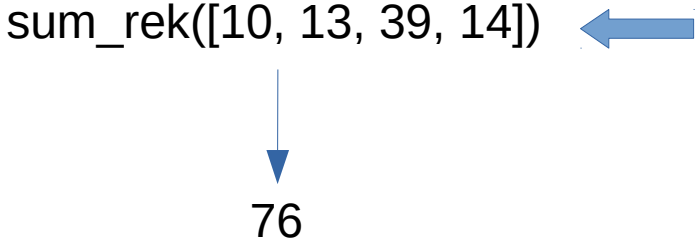
Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

Rekursionsbaum als Visualisierung




```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10         return summe
```

Warum funktioniert das?

Rekursionsbaum als Visualisierung

76 ←

```
3  def sum_rek(liste):
4      if len(liste) == 0:
5          return 0
6      else:
7          first = liste[0]
8          liste.pop(0)
9          summe = first + sum_rek(liste)
10 →       return summe
```