ETH zürich

Departement Informatik

Programmieren und Problemlösen

Binäre Suche und Rekursion

Dennis Komm

Frühling 2021 – 29. April 2021

Daten suchen

Lineare Suche

Lineare Suche

Laufe durch Liste von links nach rechts und vergleiche jedes Element mit dem gesuchten

- Einfachste Möglichkeit der Suche
- Funktioniert auch in ungeordneten Daten

Lineare Suche

Laufe durch Liste von links nach rechts und vergleiche jedes Element mit dem gesuchten

- Einfachste Möglichkeit der Suche
- Funktioniert auch in ungeordneten Daten
- Braucht bis zu n Vergleiche auf Liste der Länge n, falls gesuchtes Element an letzter Stelle ist (oder nicht vorkommt)

Lineare Suche

Laufe durch Liste von links nach rechts und vergleiche jedes Element mit dem gesuchten

- Einfachste Möglichkeit der Suche
- Funktioniert auch in ungeordneten Daten
- Braucht bis zu n Vergleiche auf Liste der Länge n, falls gesuchtes Element an letzter Stelle ist (oder nicht vorkommt)
- Komplexität in $\mathcal{O}(n)$

Lineare Suche

```
def linsearch(data, searched):
    index = 0
    while index < len(data):
        if data[index] == searched:
            return index
        index += 1
    return -1
```

Lineare Suche

```
def linsearch(data, searched):
    index = 0
    while index < len(data):
        if data[index] == searched:
            return index
        index += 1
    return -1
```

```
def linsearch(data, searched):
    if searched in data:
        return True
    else:
        return False
```

Lineare Suche

```
def linsearch(data, searched):
    index = 0
    while index < len(data):
        if data[index] == searched:
            return index
        index += 1
    return -1
```

```
def linsearch(data, searched):
    if searched in data:
        return True
    else:
        return False
```

```
def linsearch(data, searched):
    return searched in data
```

Daten suchen

Binäre Suche

Binäre Suche

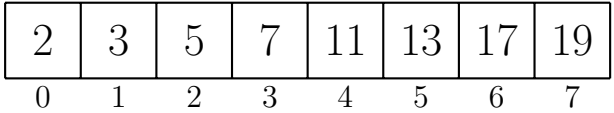
Beispiel

Gegeben eine Liste der ersten 8 Primzahlen, finde heraus, an welcher Position Primzahl 17 ist

Binäre Suche

Beispiel

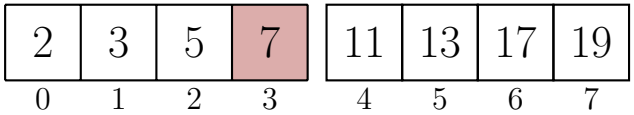
Gegeben eine Liste der ersten 8 Primzahlen, finde heraus, an welcher Position Primzahl 17 ist



Binäre Suche

Beispiel

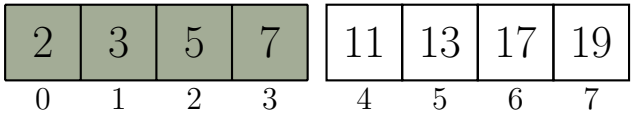
Gegeben eine Liste der ersten 8 Primzahlen, finde heraus, an welcher Position Primzahl 17 ist



Binäre Suche

Beispiel

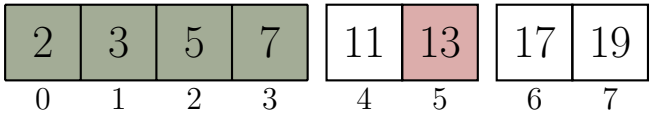
Gegeben eine Liste der ersten 8 Primzahlen, finde heraus, an welcher Position Primzahl 17 ist



Binäre Suche

Beispiel

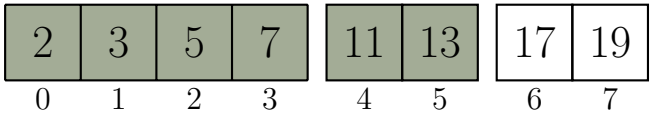
Gegeben eine Liste der ersten 8 Primzahlen, finde heraus, an welcher Position Primzahl 17 ist



Binäre Suche

Beispiel

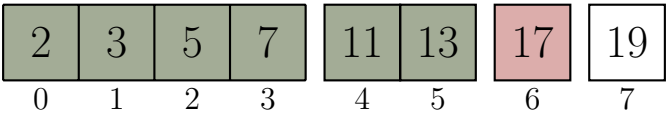
Gegeben eine Liste der ersten 8 Primzahlen, finde heraus, an welcher Position Primzahl 17 ist



Binäre Suche

Beispiel

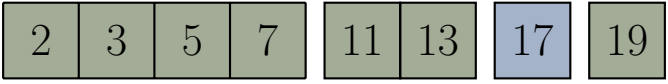
Gegeben eine Liste der ersten 8 Primzahlen, finde heraus, an welcher Position Primzahl 17 ist



Binäre Suche

Beispiel

Gegeben eine Liste der ersten 8 Primzahlen, finde heraus, an welcher Position Primzahl 17 ist



Binäre Suche

Nutze aus, dass Daten sortiert sind

Binäre Suche

Nutze aus, dass Daten sortiert sind

- Zwei Variablen `left` and `right`
- Diese geben **Suchraum** an
- Betrachte Wert in der Mitte (Index `current`)

Binäre Suche

Nutze aus, dass Daten sortiert sind

- Zwei Variablen `left` and `right`
- Diese geben **Suchraum** an
- Betrachte Wert in der Mitte (Index `current`)
- Ist dieser der gesuchte, sind wir fertig

Binäre Suche

Nutze aus, dass Daten sortiert sind

- Zwei Variablen `left` and `right`
- Diese geben **Suchraum** an
- Betrachte Wert in der Mitte (Index `current`)
- Ist dieser der gesuchte, sind wir fertig
- Ist dieser zu klein, ist auch alles links von ihm zu klein

⇨ `left = current + 1`

Binäre Suche

Nutze aus, dass Daten sortiert sind

- Zwei Variablen `left` and `right`
- Diese geben **Suchraum** an
- Betrachte Wert in der Mitte (Index `current`)
- Ist dieser der gesuchte, sind wir fertig
- Ist dieser zu klein, ist auch alles links von ihm zu klein

⇨ `left = current + 1`

- Ist dieser zu gross, ist auch alles rechts von ihm zu gross

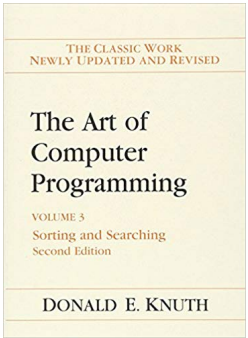
⇨ `right = current - 1`

Binäre Suche

Daten **suchen** und sortieren sind zwei der grundlegendsten Aufgaben von Informatikerinnen und Informatikern

Binäre Suche

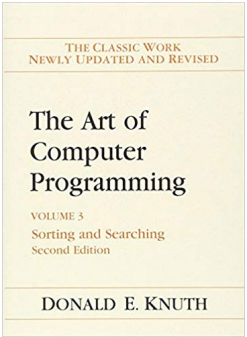
Daten **suchen** und sortieren sind zwei der grundlegendsten Aufgaben von Informatikerinnen und Informatikern



Binäre Suche

Daten **suchen** und sortieren sind zwei der grundlegendsten Aufgaben von Informatikerinnen und Informatikern

Die erste binäre Suche wurde 1946 veröffentlicht (und das Prinzip war bereits lange davor bekannt), allerdings ist die erste für alle n korrekt funktionierende Version erst 14 Jahre später erschienen



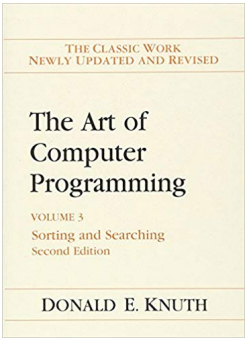
Binäre Suche

Daten **suchen** und sortieren sind zwei der grundlegendsten Aufgaben von Informatikerinnen und Informatikern

Die erste binäre Suche wurde 1946 veröffentlicht (und das Prinzip war bereits lange davor bekannt), allerdings ist die erste für alle n korrekt funktionierende Version erst 14 Jahre später erschienen

„Although the basic idea of binary search is comparatively straightforward, the details can be surprisingly tricky. . . “

–Donald Knuth



Aufgabe – Binäre Suche

Implementieren Sie die binäre Suche

- als Python-Funktion
- mit „Zeigern“ `left`, `right` und `current`
- Am Anfang ist `left = 0` und `right = len(data) - 1`
- Verkleinern Sie den Suchraum in jedem Schritt wie beschrieben
- Wenn Element gefunden wird, wird seine Position zurückgegeben
- Sonst wird -1 zurückgegeben

Binäre Suche

```
def binsearch(data, searched):
    left = 0
    right = len(data) - 1
    while left <= right:
        current = (left + right) // 2
        if data[current] == searched:
            return current
        elif data[current] > searched:
            right = current - 1
        else:
            left = current + 1
    return -1
```

Suchen

Komplexität binärer Suche

Komplexität binärer Suche

- Am Anfang hat Liste n Elemente

Komplexität binärer Suche

- Am Anfang hat Liste n Elemente
- Mit jeder Iteration wird der **Suchraum** auf die Hälfte reduziert

Komplexität binärer Suche

- Am Anfang hat Liste n Elemente
- Mit jeder Iteration wird der **Suchraum** auf die Hälfte reduziert
- Nach der ersten Iteration $n/2$ Elemente

Komplexität binärer Suche

- Am Anfang hat Liste n Elemente
- Mit jeder Iteration wird der **Suchraum** auf die Hälfte reduziert
- Nach der ersten Iteration $n/2$ Elemente
- Nach der zweiten Iteration $n/4$ Elemente

Komplexität binärer Suche

- Am Anfang hat Liste n Elemente
- Mit jeder Iteration wird der **Suchraum** auf die Hälfte reduziert
- Nach der ersten Iteration $n/2$ Elemente
- Nach der zweiten Iteration $n/4$ Elemente
- ...

Komplexität binärer Suche

- Am Anfang hat Liste n Elemente
- Mit jeder Iteration wird der **Suchraum** auf die Hälfte reduziert
- Nach der ersten Iteration $n/2$ Elemente
- Nach der zweiten Iteration $n/4$ Elemente
- ...
- Nach wie vielen Iterationen x ist nur noch ein Element übrig?

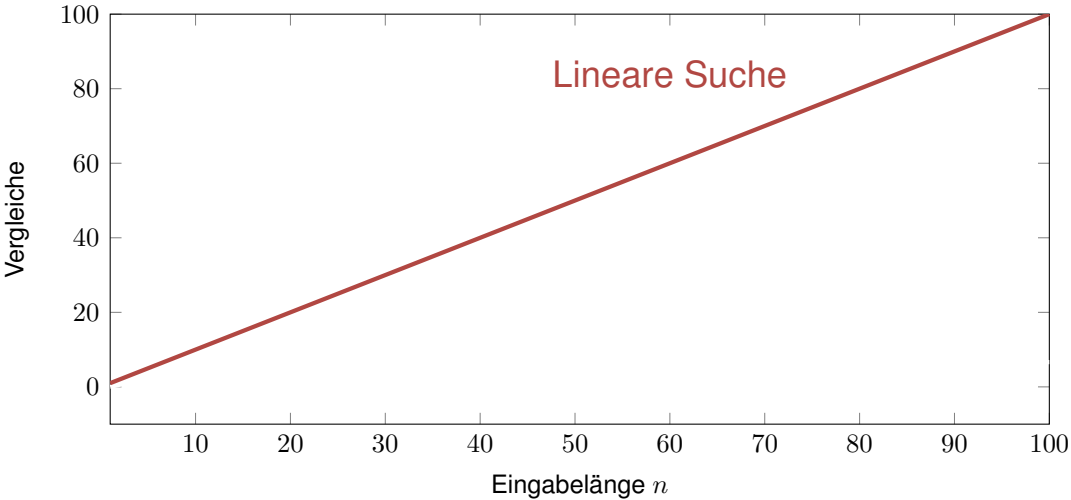
Komplexität binärer Suche

- Am Anfang hat Liste n Elemente
- Mit jeder Iteration wird der **Suchraum** auf die Hälfte reduziert
- Nach der ersten Iteration $n/2$ Elemente
- Nach der zweiten Iteration $n/4$ Elemente
- ...
- Nach wie vielen Iterationen x ist nur noch ein Element übrig?
- $n/2^x = 1 \iff n = 2^x \iff x = \log_2 n$

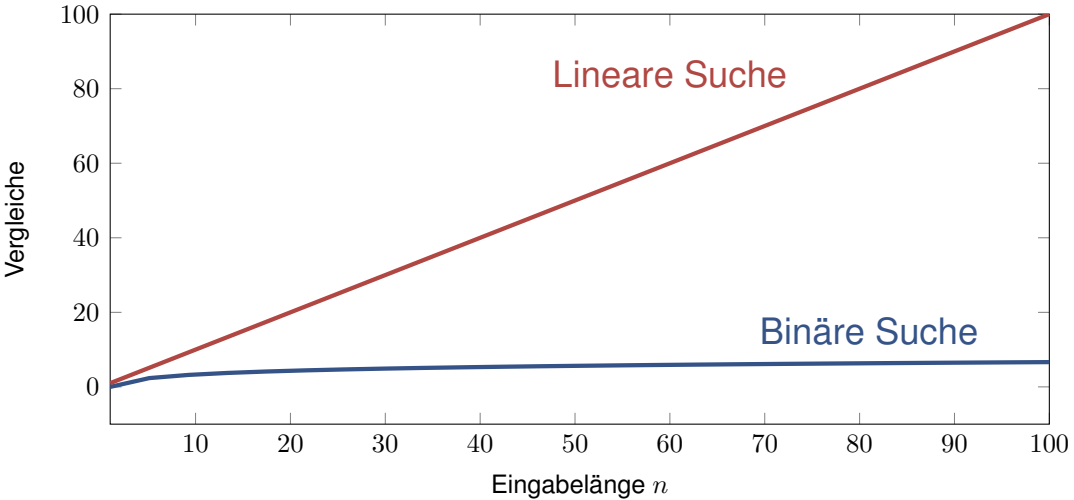
Komplexität binärer Suche

- Am Anfang hat Liste n Elemente
- Mit jeder Iteration wird der **Suchraum** auf die Hälfte reduziert
- Nach der ersten Iteration $n/2$ Elemente
- Nach der zweiten Iteration $n/4$ Elemente
- ...
- Nach wie vielen Iterationen x ist nur noch ein Element übrig?
- $n/2^x = 1 \iff n = 2^x \iff x = \log_2 n$
- Komplexität in $\mathcal{O}(\log_2 n)$

Komplexität binärer Suche



Komplexität binärer Suche



Komplexität binärer Suche

Laufzeit der binären Suche im schlechtesten Fall darstellen

- Wir verwenden wieder eine Variable `counter`, um Vergleiche zu zählen
- Algorithmus wird auf sortierten Listen mit den Werten 1 bis n laufen gelassen
- Der Wert von n wächst mit jeder Iteration um 1
- Am Anfang ist n dabei 1, am Ende $1\,000\,000$
- Gesucht wird immer das erste Element 1
- Ergebnis wird in einer Liste gespeichert und mit `matplotlib` geplottet

Komplexität binärer Suche

Worst Case

```
values = []
data = [1]


for i in range(1, 1000001):
    data.append(data[-1] + 1)
    values.append(binsearch(data, 1))


plt.plot(values, color="red")
plt.show()
```

Komplexität binärer Suche

Worst Case

```
values = []
data = [1]


for i in range(1, 1000001):
    data.append(data[-1] + 1)
    values.append(binsearch(data, 1))


plt.plot(values, color="red")
plt.show()
```

Füge Element hinzu,
das eins grösser ist
als das aktuell letzte

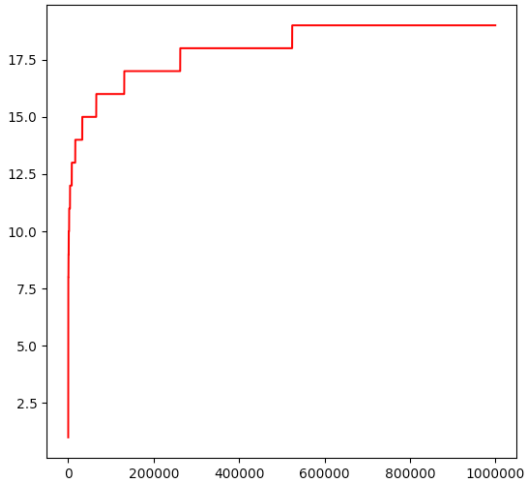
Komplexität binärer Suche

Worst Case

```
values = []
data = [1]


for i in range(1, 1000001):
    data.append(data[-1] + 1)
    values.append(binsearch(data, 1))


plt.plot(values, color="red")
plt.show()
```



Komplexität binärer Suche

Was, wenn Daten unsortiert sind?

Komplexität binärer Suche

Was, wenn Daten unsortiert sind?

- Lineare Suche funktioniert auch in unsortierten Listen und ist in $\mathcal{O}(n)$

Komplexität binärer Suche

Was, wenn Daten unsortiert sind?

- Lineare Suche funktioniert auch in unsortierten Listen und ist in $\mathcal{O}(n)$
- Bei mehrfacher Suche kann einmaliges Sortieren sich lohnen
- Sortieren ist in $\mathcal{O}(n \log_2 n)$ und damit langsamer als lineare Suche
- Binäre Suche ist in $\mathcal{O}(\log_2 n)$ und damit allerdings wiederum viel schneller als lineare Suche

Komplexität binärer Suche

Was, wenn Daten unsortiert sind?

- Lineare Suche funktioniert auch in unsortierten Listen und ist in $\mathcal{O}(n)$
- Bei mehrfacher Suche kann einmaliges Sortieren sich lohnen
- Sortieren ist in $\mathcal{O}(n \log_2 n)$ und damit langsamer als lineare Suche
- Binäre Suche ist in $\mathcal{O}(\log_2 n)$ und damit allerdings wiederum viel schneller als lineare Suche

Wann lohnt sich Sortieren?

Komplexität binärer Suche

Was, wenn Daten unsortiert sind?

- Lineare Suche funktioniert auch in unsortierten Listen und ist in $\mathcal{O}(n)$
- Bei mehrfacher Suche kann einmaliges Sortieren sich lohnen
- Sortieren ist in $\mathcal{O}(n \log_2 n)$ und damit langsamer als lineare Suche
- Binäre Suche ist in $\mathcal{O}(\log_2 n)$ und damit allerdings wiederum viel schneller als lineare Suche

Wann lohnt sich Sortieren?

Wenn öfter als $\log_2 n$ Mal gesucht werden muss

Rekursive Funktionen

Rekursive Funktionen

`def f():` $\Longleftrightarrow$ Python „lernt“ neues Wort `f`

Rekursive Funktionen

`def f():` $\Longleftrightarrow$ Python „lernt“ neues Wort `f`

Aus dem Duden

Kühl·schrank, der

Mit einer Kältemaschine ausgestatteter schrankartiger Behälter zum Kühlen oder Frischhalten von Lebensmitteln

Rekursive Funktionen

`def f():` $\Longleftrightarrow$ Python „lernt“ neues Wort `f`

Aus dem Duden

Kühl·schrank, der

Mit einer Kältemaschine ausgestatteter schrankartiger Behälter zum Kühlen oder Frischhalten von Lebensmitteln

- Diese Analogie stimmt nicht ganz, denn Funktionen können „sich selber“ aufrufen
- Solche Funktionen heissen **rekursive Funktionen**

Rekursive Funktionen

`def f():` $\Longleftrightarrow$ Python „lernt“ neues Wort `f`

Aus dem Duden

Kühl·schrank, der
Mit einer Kältemaschine ausgestatteter schrankartiger Behälter zum Kühlen oder Frischhalten von Lebensmitteln

- Diese Analogie stimmt nicht ganz, denn Funktionen können „sich selber“ aufrufen
- Solche Funktionen heissen **rekursive Funktionen**

Nicht aus dem Duden

Kühl·schrank, der
Ein Kühlschrank

Rekursive Funktionen

Dies führt zunächst einmal zu einer **Endlosschleife**

Rekursive Funktionen

Dies führt zunächst einmal zu einer **Endlosschleife**

```
def f():
    print("Hallo, Welt!")
    f()
```

Rekursive Funktionen

Dies führt zunächst einmal zu einer **Endlosschleife**

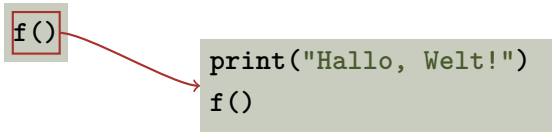
```
def f():
    print("Hallo, Welt!")
    f()
```

```
f()
```

Rekursive Funktionen

Dies führt zunächst einmal zu einer **Endlosschleife**

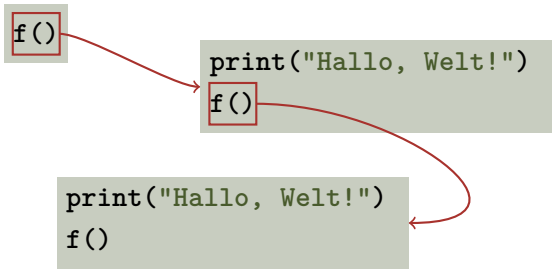
```
def f():
    print("Hallo, Welt!")
    f()
```



Rekursive Funktionen

Dies führt zunächst einmal zu einer **Endlosschleife**

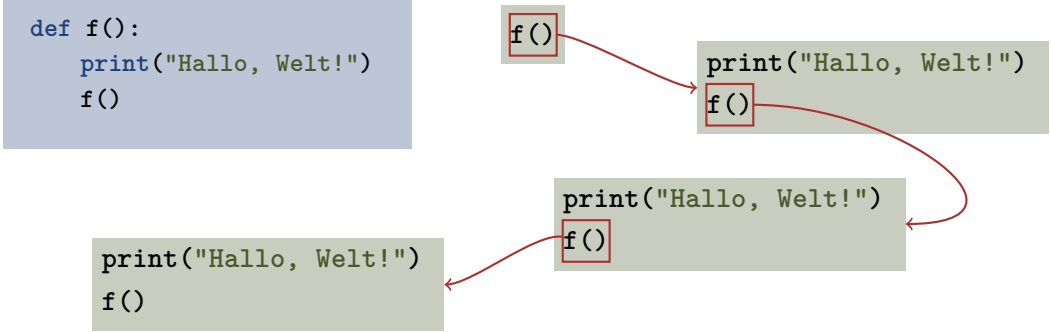
```
def f():
    print("Hallo, Welt!")
    f()
```



Rekursive Funktionen

Dies führt zunächst einmal zu einer **Endlosschleife**

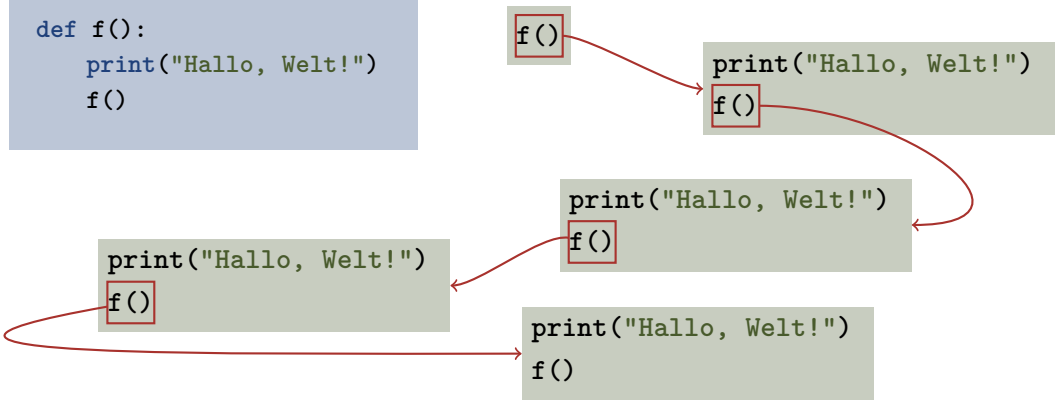
```
def f():
    print("Hallo, Welt!")
    f()
```



Rekursive Funktionen

Dies führt zunächst einmal zu einer **Endlosschleife**

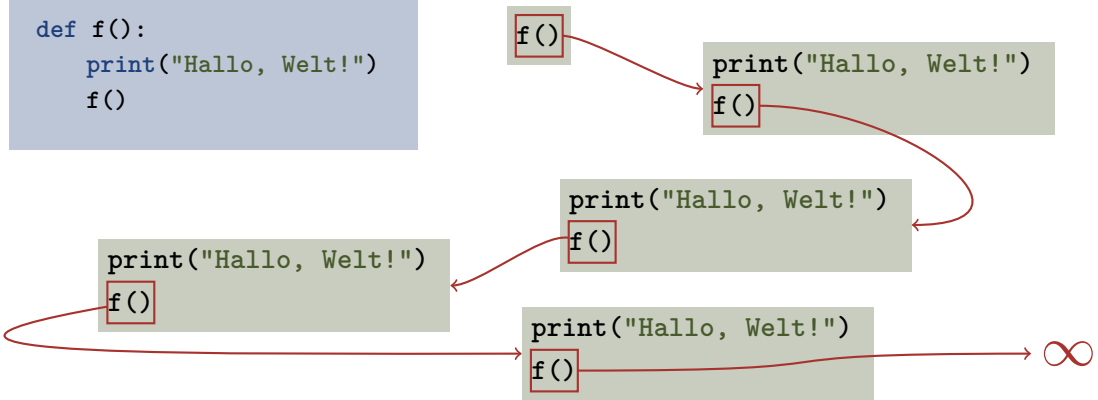
```
def f():
    print("Hallo, Welt!")
    f()
```



Rekursive Funktionen

Dies führt zunächst einmal zu einer **Endlosschleife**

```
def f():
    print("Hallo, Welt!")
    f()
```



Rekursive Funktionen

Wir verwenden Parameter, um nach endlicher Zahl von Aufrufen abzubrechen

Rekursive Funktionen

Wir verwenden Parameter, um nach endlicher Zahl von Aufrufen abzubrechen

```
def f(k):
    print(k)
    if k == 1:
        return
    else:
        f(k-1)
```

Rekursive Funktionen

Wir verwenden Parameter, um nach endlicher Zahl von Aufrufen abzubrechen

```
def f(k):
    print(k)
    if k == 1:
        return
    else:
        f(k-1)
```

Parameter (oder jedwede lokale Variable)
wird für jeden Aufruf neu erstellt

Rekursive Funktionen

Wir verwenden Parameter, um nach endlicher Zahl von Aufrufen abzubrechen

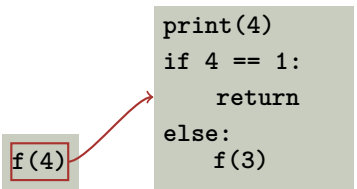
```
def f(k):
    print(k)
    if k == 1:
        return
    else:
        f(k-1)
```

```
f(4)
```

Rekursive Funktionen

Wir verwenden Parameter, um nach endlicher Zahl von Aufrufen abzubrechen

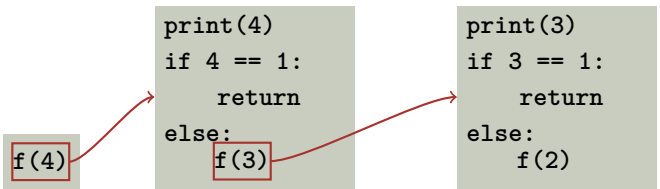
```
def f(k):
    print(k)
    if k == 1:
        return
    else:
        f(k-1)
```



Rekursive Funktionen

Wir verwenden Parameter, um nach endlicher Zahl von Aufrufen abzubrechen

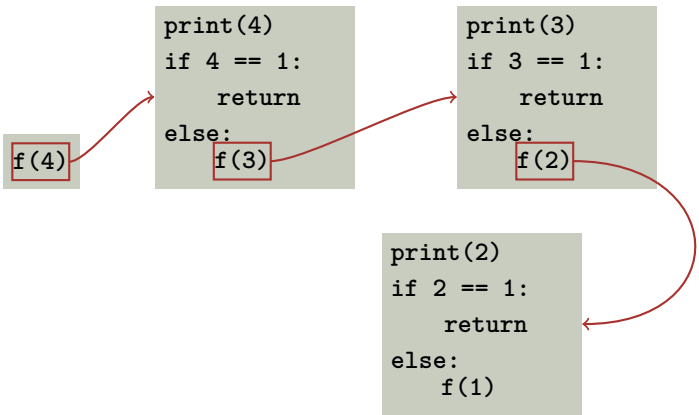
```
def f(k):
    print(k)
    if k == 1:
        return
    else:
        f(k-1)
```



Rekursive Funktionen

Wir verwenden Parameter, um nach endlicher Zahl von Aufrufen abzubrechen

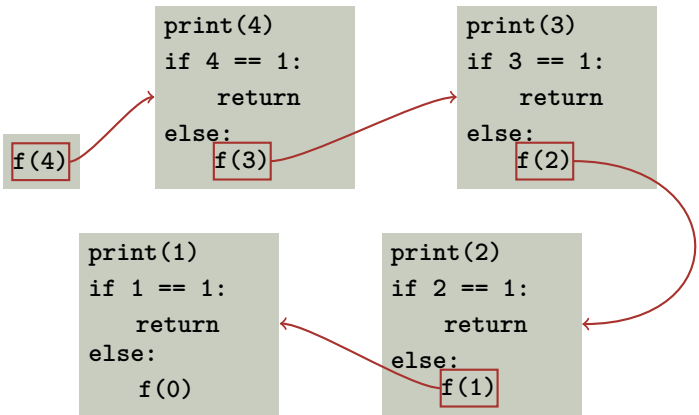
```
def f(k):
    print(k)
    if k == 1:
        return
    else:
        f(k-1)
```



Rekursive Funktionen

Wir verwenden Parameter, um nach endlicher Zahl von Aufrufen abzubrechen

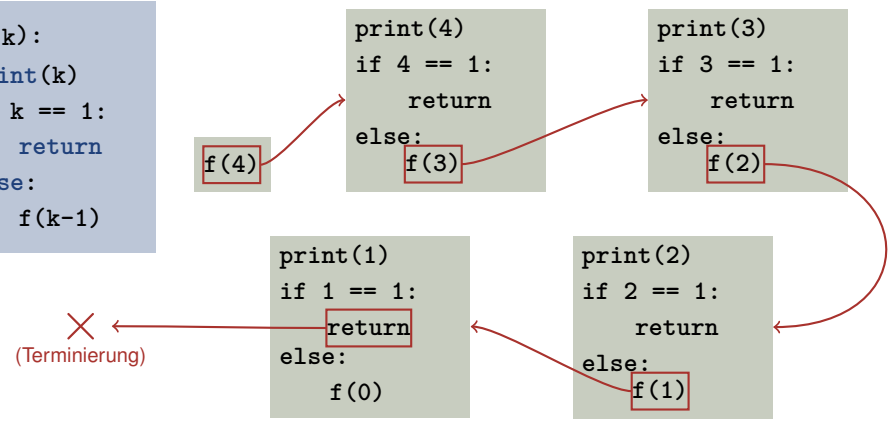
```
def f(k):
    print(k)
    if k == 1:
        return
    else:
        f(k-1)
```



Rekursive Funktionen

Wir verwenden Parameter, um nach endlicher Zahl von Aufrufen abzubrechen

```
def f(k):
    print(k)
    if k == 1:
        return
    else:
        f(k-1)
```



Fakultät und Summe

Fakultät rekursiv berechnen

- Fakultät einer natürlichen Zahl n ist definiert als

$$\text{fact}(n) = n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1$$

Fakultät rekursiv berechnen

- Fakultät einer natürlichen Zahl n ist definiert als

$$\text{fact}(n) = n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1$$

- Zum Beispiel ist $7! = 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5040$

Fakultät rekursiv berechnen

- Fakultät einer natürlichen Zahl n ist definiert als

$$\text{fact}(n) = n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1$$

- Zum Beispiel ist $7! = 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5040$
- Wir beobachten

$$n! = n \cdot (n-1)! = n \cdot (n-1) \cdot (n-2)! = \dots$$

Fakultät rekursiv berechnen

- Fakultät einer natürlichen Zahl n ist definiert als

$$\text{fact}(n) = n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1$$

- Zum Beispiel ist $7! = 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5040$
- Wir beobachten

$$n! = n \cdot (n-1)! = n \cdot (n-1) \cdot (n-2)! = \dots$$

- Funktion kann rekursiv berechnet werden mit

$$\text{fact}(1) = 1 \quad \text{und} \quad \text{fact}(n) = n \cdot \text{fact}(n-1)$$

Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Wie vorher Parameter, der bei jedem Aufruf neu erstellt wird

Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Letzter Aufruf gibt
festen Wert 1 zurück

Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Sonstige Aufrufe geben
n Mal „Fakultät von `n-1`“ zurück

Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

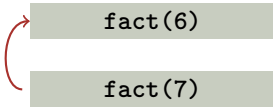
Aufruf-Stack

`fact(7)`

Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

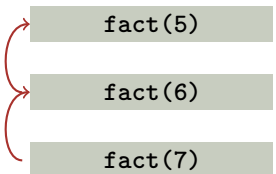
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

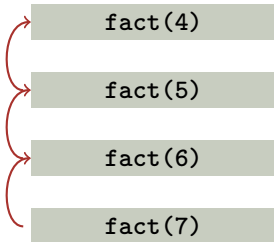
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

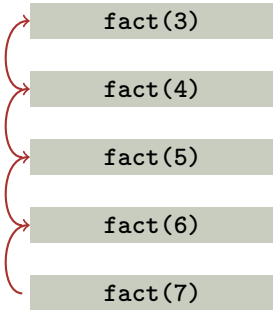
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

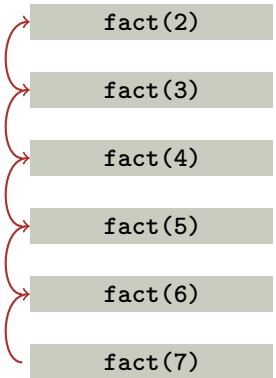
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

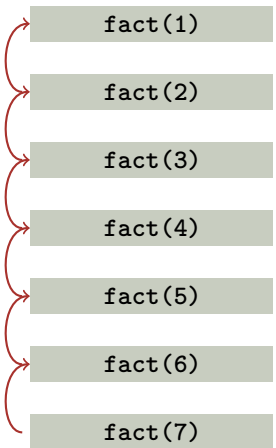
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

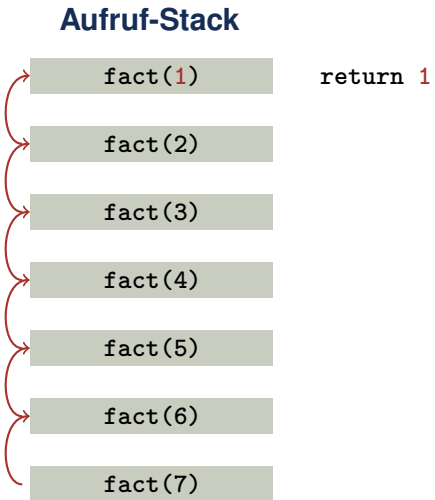
```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Aufruf-Stack



Fakultät rekursiv berechnen – in Python

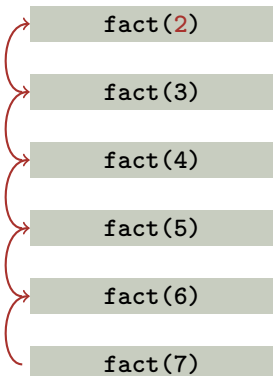
```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

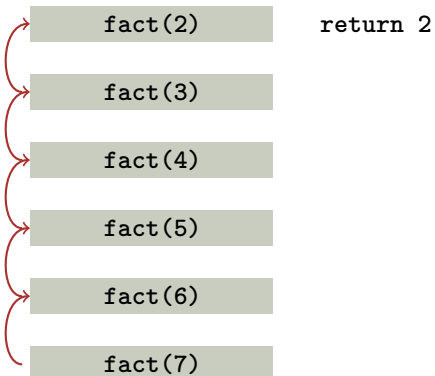
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

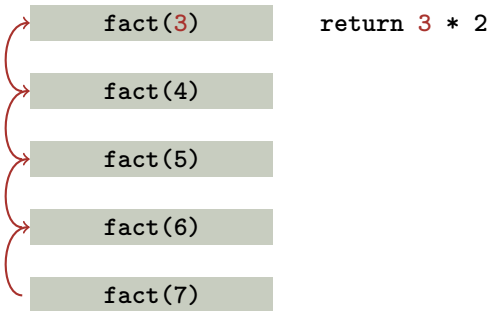
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

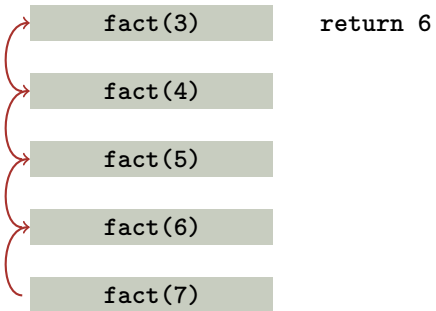
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

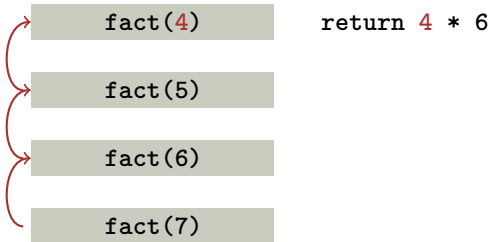
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

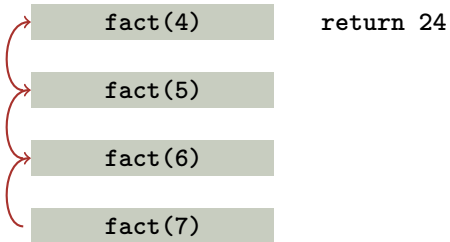
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

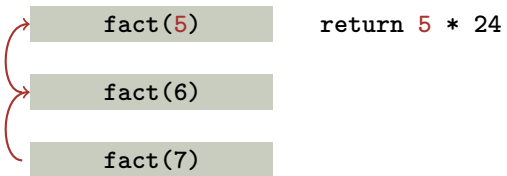
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

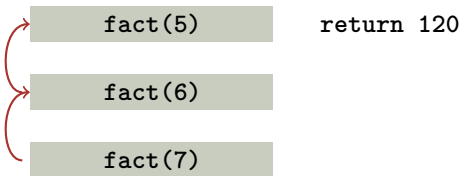
Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Aufruf-Stack



Fakultät rekursiv berechnen – in Python

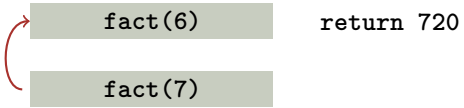
```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Aufruf-Stack

Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Aufruf-Stack



Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Aufruf-Stack

`fact(7)`

`return 7 * 720`

Fakultät rekursiv berechnen – in Python

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Aufruf-Stack

`fact(7)` `return 5040`

Aufgabe – Summe rekursiv berechnen

Implementieren Sie eine rekursive Funktion, die

- einen Parameter `n` erhält
- und die Summe der ersten `n` natürlichen Zahlen zurückgibt

Aufgabe – Summe rekursiv berechnen

Beide rekursive Funktionen können mit derselben Idee umgesetzt werden

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)
```

```
def thesum(n):
    if n == 1:
        return 1
    else:
        return n + thesum(n-1)
```

Rekursion vs. Iteration

- Es existieren Alternativen mit Schleifen

```
def fact(n):
    i = 1
    result = 1
    while i < n:
        i += 1
        result *= i
    return result
```

```
def thesum(n):
    i = 1
    result = 1
    while i < n:
        i += 1
        result += i
    return result
```

Rekursion vs. Iteration

- Es existieren Alternativen mit Schleifen

```
def fact(n):
    i = 1
    result = 1
    while i < n:
        i += 1
        result *= i
    return result
```

```
def thesum(n):
    i = 1
    result = 1
    while i < n:
        i += 1
        result += i
    return result
```

- Für die Summe existiert ausserdem eine geschlossene Form („der kleine Gauss“ aus der Bubblesort-Analyse)

```
def thesum(n):
    return n * (n+1) / 2
```

Rekursion vs. Iteration

Wenn sich wiederholende Anweisungen durch Schleifen ausgeführt werden, reden wir von **iterativer Programmierung**

Rekursion vs. Iteration

Wenn sich wiederholende Anweisungen durch Schleifen ausgeführt werden, reden wir von **iterativer Programmierung**

- Für alle Probleme gibt es iterative und rekursive Lösungen

Rekursion vs. Iteration

Wenn sich wiederholende Anweisungen durch Schleifen ausgeführt werden, reden wir von **iterativer Programmierung**

- Für alle Probleme gibt es iterative und rekursive Lösungen
- Oftmals kann die rekursive Lösung als „eleganter“ betrachtet werden

Rekursion vs. Iteration

Wenn sich wiederholende Anweisungen durch Schleifen ausgeführt werden, reden wir von **iterativer Programmierung**

- Für alle Probleme gibt es iterative und rekursive Lösungen
- Oftmals kann die rekursive Lösung als „eleganter“ betrachtet werden
- Die Umsetzung mit Rekursion ist oft kürzer (prägnanter) zu schreiben
- . . . aber selten schneller auszuführen

Rekursion vs. Iteration

Wenn sich wiederholende Anweisungen durch Schleifen ausgeführt werden, reden wir von **iterativer Programmierung**

- Für alle Probleme gibt es iterative und rekursive Lösungen
- Oftmals kann die rekursive Lösung als „eleganter“ betrachtet werden
- Die Umsetzung mit Rekursion ist oft kürzer (prägnanter) zu schreiben
- . . . aber selten schneller auszuführen
- Was verwendet werden sollte, hängt von diversen Faktoren ab

Rekursiver Euklidischer Algorithmus

Rekursiver Euklidischer Algorithmus

Euklidischer Algorithmus

bekannt aus der ersten Vorlesung

- Eingabe: ganze Zahlen $a > 0$, $b > 0$
- Ausgabe: ggT von a und b

```
def euclid(a, b):
    while b != 0:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```

Rekursiver Euklidischer Algorithmus

Euklidischer Algorithmus

bekannt aus der ersten Vorlesung

- Eingabe: ganze Zahlen $a > 0, b > 0$
- Ausgabe: ggT von a und b

```
def euclid(a, b):
    while b != 0:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```

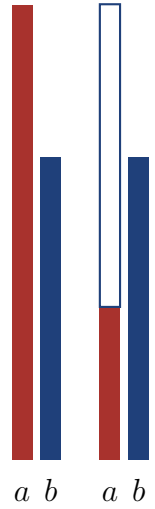
Rekursiver Euklidischer Algorithmus

Euklidischer Algorithmus

bekannt aus der ersten Vorlesung

- Eingabe: ganze Zahlen $a > 0, b > 0$
- Ausgabe: ggT von a und b

```
def euclid(a, b):
    while b != 0:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```



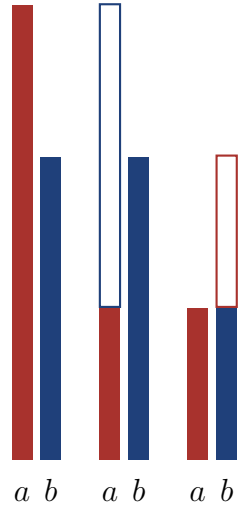
Rekursiver Euklidischer Algorithmus

Euklidischer Algorithmus

bekannt aus der ersten Vorlesung

- Eingabe: ganze Zahlen $a > 0, b > 0$
- Ausgabe: ggT von a und b

```
def euclid(a, b):
    while b != 0:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```



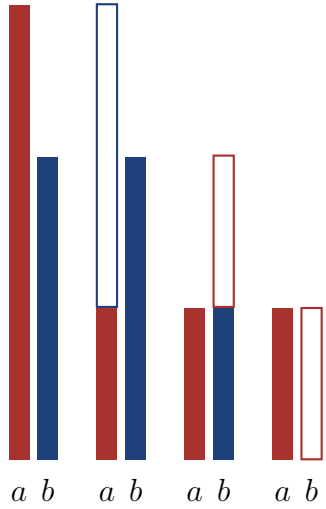
Rekursiver Euklidischer Algorithmus

Euklidischer Algorithmus

bekannt aus der ersten Vorlesung

- Eingabe: ganze Zahlen $a > 0, b > 0$
- Ausgabe: ggT von a und b

```
def euclid(a, b):
    while b != 0:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```



Aufgabe – GGT rekursiv berechnen

Implementieren Sie den Euklidischen Algorithmus

- als rekursive Python-Funktion
- mit zwei Parametern `a` and `b`

```
def euclid(a, b):
    while b != 0:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```

GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

```
def euclid(a, b):
    while b != 0:



        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```

GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

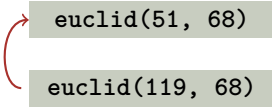
`euclid(119, 68)`

GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

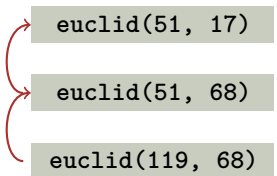


GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

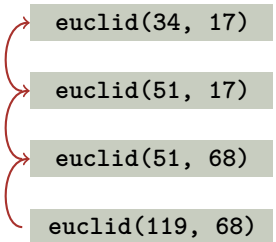


GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

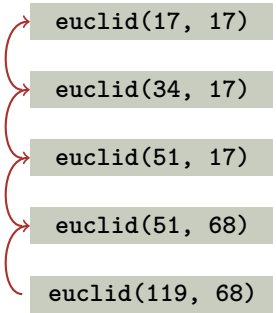


GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

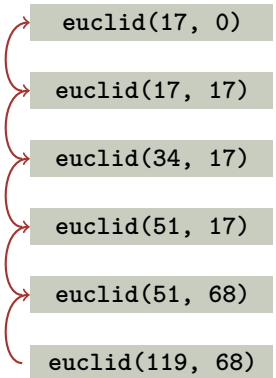


GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

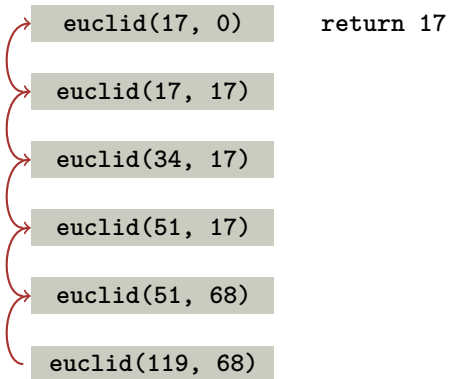


GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

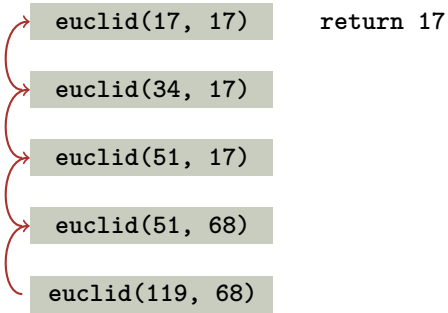


GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

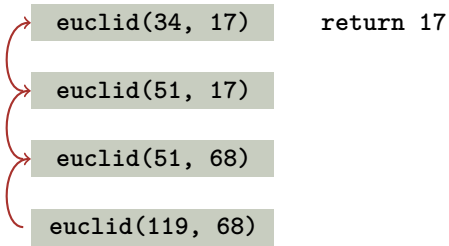


GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

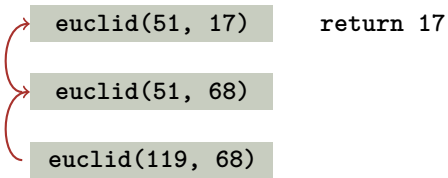


GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

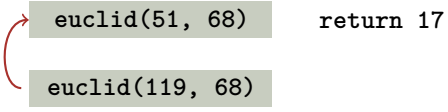


GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht



GGT rekursiv berechnen

```
def euclid(a, b):
    if b == 0:
        return a
    else:
        if a > b:
            return euclid(a - b, b)
        else:
            return euclid(a, b - a)
```

Aufruf-Stack

`return`-Wert wird direkt durchgereicht

`euclid(119, 68)` `return 17`

Rekursives Sortieren und Suchen

Binäre Suche

Iterative Binäre Suche

```
def binsearch(data, searched):
    left = 0
    right = len(data) - 1
    while left <= right:
        current = (left + right) // 2
        if data[current] == searched:
            return current
        elif data[current] > searched:
            right = current - 1
        else:
            left = current + 1
    return -1
```

Rekursive Binäre Suche

Rekursive Implementierung

- Funktion besitzt wieder Parameter `data` und `searched` für die gegebene Liste und das gesuchte Element

Rekursive Binäre Suche

Rekursive Implementierung

- Funktion besitzt wieder Parameter `data` und `searched` für die gegebene Liste und das gesuchte Element
- Zwei weitere Parameter `left` und `right` geben aktuellen **Suchraum** an
- In einem einzelnen Aufruf werden `left` und `right` nicht verändert

⇨ **Keine** Schleife

Rekursive Binäre Suche

Rekursive Implementierung

- Funktion besitzt wieder Parameter `data` und `searched` für die gegebene Liste und das gesuchte Element
- Zwei weitere Parameter `left` und `right` geben aktuellen **Suchraum** an
- In einem einzelnen Aufruf werden `left` und `right` nicht verändert

⇨ **Keine** Schleife

- `current` wird wieder berechnet als `(left + right) // 2`
- Betrachte wieder Stelle `data[current]`

Rekursive Binäre Suche

Rekursive Implementierung

- Funktion besitzt wieder Parameter `data` und `searched` für die gegebene Liste und das gesuchte Element
- Zwei weitere Parameter `left` und `right` geben aktuellen **Suchraum** an
- In einem einzelnen Aufruf werden `left` und `right` nicht verändert

⇨ **Keine** Schleife

- `current` wird wieder berechnet als `(left + right) // 2`
- Betrachte wieder Stelle `data[current]`
- Wenn `searched` nicht gefunden wurde, rufe Funktion auf und passe entweder `left` oder `right` entsprechend an

Aufgabe – Rekursive Binäre Suche

Implementieren Sie die binäre Suche

- als rekursive Python-Funktion
- mit vier Parametern
  `data`, `left`, `right` und `searched`
- Orientieren Sie sich an der iterativen Variante

```
def binsearch(data, searched):
    left = 0
    right = len(data) - 1
    while left <= right:
        current = (left + right) // 2
        if data[current] == searched:
            return current
        elif data[current] > searched:
            right = current - 1
        else:
            left = current + 1
    return -1
```

Rekursive Binäre Suche

```
def binsearch(data, left, right, searched):


    if left <= right:
        current = (left + right) // 2
        if data[current] == searched:
            return current
        elif data[current] > searched:
            return binsearch(data, left, current-1, searched)
        else:
            return binsearch(data, current+1, right, searched)
    else:
        return -1
```

Rekursive Binäre Suche

```
def binsearch(data, left, right, searched):


    if left <= right:
        current = (left + right) // 2
        if data[current] == searched:
            return current
        elif data[current] > searched:
            return binsearch(data, left, current-1, searched)
        else:
            return binsearch(data, current+1, right, searched)
    else:
        return -1
```

```
def binsearch(data, searched):
    left = 0
    right = len(data) - 1
    while left <= right:
        current = (left + right) // 2
        if data[current] == searched:
            return current
        elif data[current] > searched:
            right = current - 1
        else:
            left = current + 1

    return -1
```

Rekursive Binäre Suche

Aufruf-Stack

`return`-Wert wird wieder direkt durchgereicht

Rekursive Binäre Suche

Aufruf-Stack

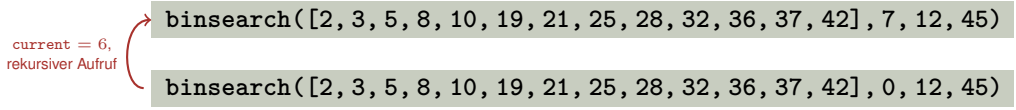
`return`-Wert wird wieder direkt durchgereicht

```
binsearch([2, 3, 5, 8, 10, 19, 21, 25, 28, 32, 36, 37, 42], 0, 12, 45)
```

Rekursive Binäre Suche

Aufruf-Stack

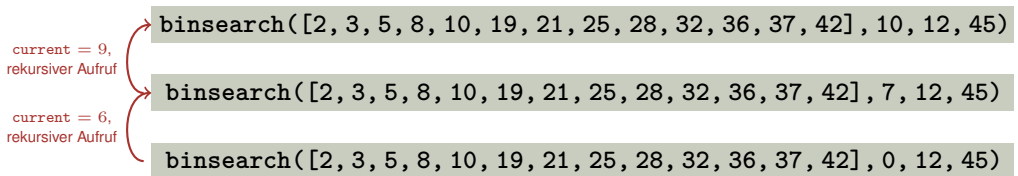
`return`-Wert wird wieder direkt durchgereicht



Rekursive Binäre Suche

Aufruf-Stack

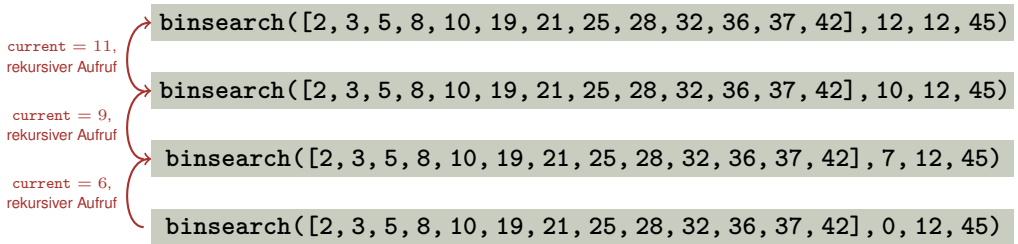
`return`-Wert wird wieder direkt durchgereicht



Rekursive Binäre Suche

Aufruf-Stack

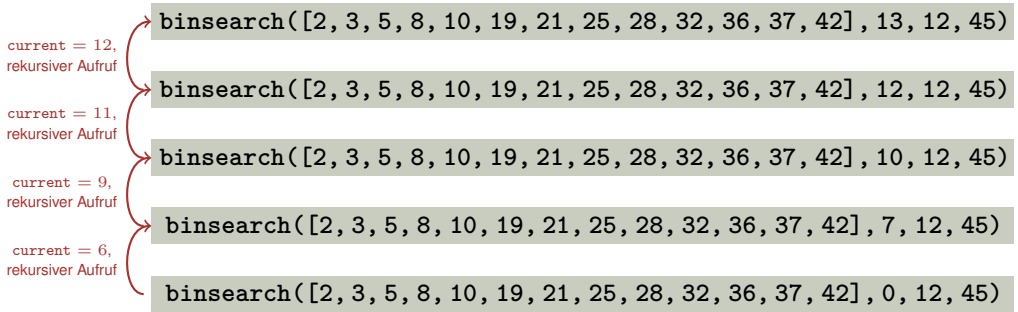
`return`-Wert wird wieder direkt durchgereicht



Rekursive Binäre Suche

Aufruf-Stack

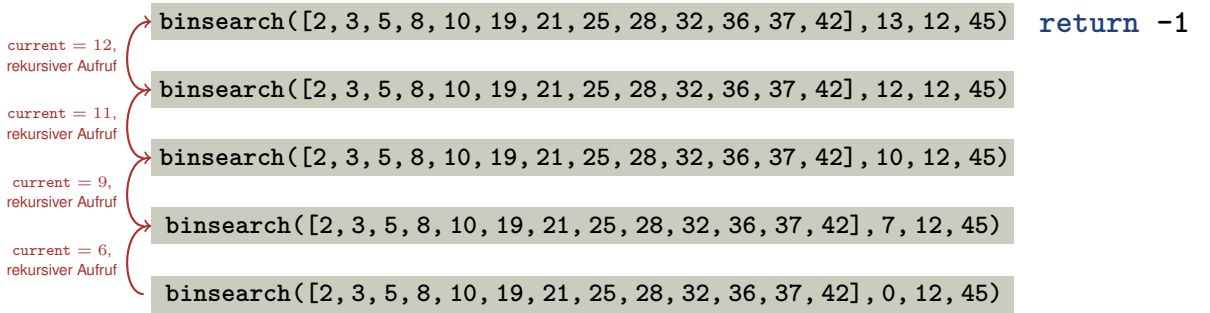
`return`-Wert wird wieder direkt durchgereicht



Rekursive Binäre Suche

Aufruf-Stack

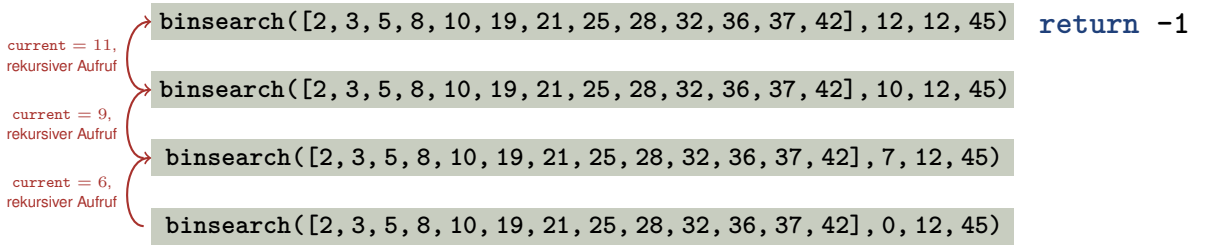
`return`-Wert wird wieder direkt durchgereicht



Rekursive Binäre Suche

Aufruf-Stack

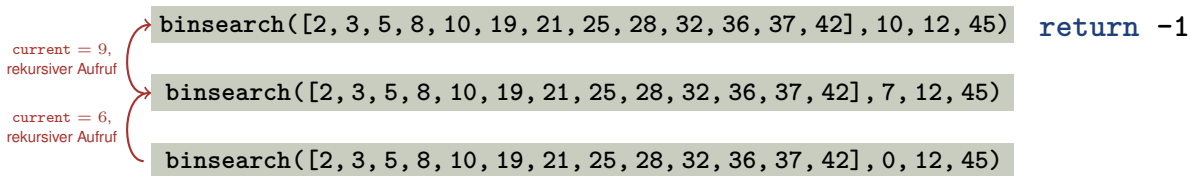
`return`-Wert wird wieder direkt durchgereicht



Rekursive Binäre Suche

Aufruf-Stack

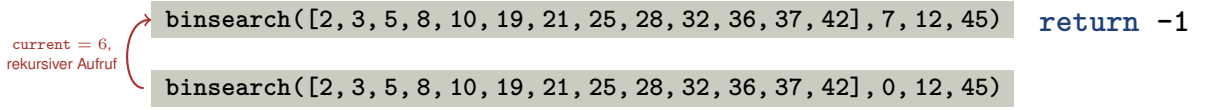
`return`-Wert wird wieder direkt durchgereicht



Rekursive Binäre Suche

Aufruf-Stack

`return`-Wert wird wieder direkt durchgereicht



Rekursive Binäre Suche

Aufruf-Stack

`return`-Wert wird wieder direkt durchgereicht

```
binsearch([2,3,5,8,10,19,21,25,28,32,36,37,42],0,12,45)  return -1
```

Danke für die Aufmerksamkeit