Rekursion 3

D. Komm, J. Závodný

26.02.2020

Listen

Man hat eine Liste.

```
a = [5, 7, 32, 4, 12]
```

Wie bekommt man eine Liste, in der jedes Element um 1 höher ist?

Listen

Man hat eine Liste.

```
a = [5, 7, 32, 4, 12]
```

Wie bekommt man eine Liste, in der jedes Element um 1 höher ist?

```
b = []
for x in a:
  b.append(x + 1)
```

Listen

Man hat eine Liste.

```
a = [5, 7, 32, 4, 12]
```

Wie bekommt man eine Liste, in der jedes Element um 1 höher ist?

```
b = []
for x in a:
  b.append(x + 1)
```

```
c = [x + 1 for x in a]
```

Listen

Man hat eine Liste.

```
a = [5, 7, 32, 4, 12]
```

Wie bekommt man eine Liste mit den Quadraten aller Elemente in der Liste?

Listen

Man hat eine Liste.

```
a = [5, 7, 32, 4, 12]
```

Wie bekommt man eine Liste mit den Quadraten aller Elemente in der Liste?

```
b = []
for x in a:
  b.append(x * x)
```

Listen

Man hat eine Liste.

```
a = [5, 7, 32, 4, 12]
```

Wie bekommt man eine Liste mit den Quadraten aller Elemente in der Liste?

```
b = []
for x in a:
  b.append(x * x)

c = [x * x for x in a]
```

Listen

Man hat eine Liste.

```
a = [5, 2, 7, 33, 4, 12]
```

Wie bekommt man eine Liste mit allen geraden Zahlen in der Liste?

Listen

Man hat eine Liste.

```
a = [5, 2, 7, 33, 4, 12]
```

Wie bekommt man eine Liste mit allen geraden Zahlen in der Liste?

```
b = []
for x in a:
  if x % 2 == 0:
    b.append(x)
```

Listen

Man hat eine Liste.

```
a = [5, 2, 7, 33, 4, 12]
```

Wie bekommt man eine Liste mit allen geraden Zahlen in der Liste?

```
b = []
for x in a:
  if x % 2 == 0:
    b.append(x)

c = [x for x in a if x % 2 == 0]
```

Listen

Man hat eine Liste mit Englischen Wörtern.

```
a = ["dog", "cat", "bird"]
```

Wie bekommt man eine Liste mit den Pluralformen?

Listen

Man hat eine Liste mit Englischen Wörtern.

```
a = ["dog", "cat", "bird"]
```

Wie bekommt man eine Liste mit den Pluralformen?

```
b = []
for w in a:
  b.append(w + "s")

c = [w + "s" for w in a]
```

Listen

Man hat eine Liste mit Englischen Wörtern.

```
a = ["dog", "cat", "bird",  "octopus", "fox"]
```

Wie bekommt man eine Liste mit den Pluralformen?

```
b = []
for w in a:
  b.append(w + "s")

> ["dogs", "cats", "birds",  "octopuss", "foxs"]
```

Listen

Man hat eine Liste mit Englischen Wörtern.

```
a = ["dog", "cat", "bird",  "octopus", "fox"]
```

Wie bekommt man eine Liste mit den Pluralformen?

```
b = []
for w in a:
  if s[-1] == "s" or s[-1] == "x":
    b.append(w + "es")
  else:
    b.append(w + "s")
```

Listen

Man hat eine Liste mit Englischen Wörtern.

```
a = ["dog", "cat", "bird", "octopus", "fox"]
```

Wie bekommt man eine Liste mit den Pluralformen?

```
b = []
for w in a:
  if s[-1] == "s" or s[-1] == "x":
    b.append(w + "es")
  else:
    b.append(w + "s")
```

```
c = [(w + "es" if (s[-1] == "s" or s[-1] == "x") else w + "s") for w in a]
```

Listen

Man hat eine Liste mit Englischen Wörtern.

```
a = ["dog", "cat", "bird", "octopus", "fox",  "fish"]
```

Wie bekommt man eine Liste mit den Pluralformen?

```
b = []
for w in a:
  if s[-1] == "s" or s[-1] == "x":
    b.append(w + "es")
  else:
    b.append(w + "s")

c = [(w + "es" if (s[-1] == "s" or s[-1] == "x") else w + "s") for w in a]
```

Rekursion mit Listen! (Warm-up)

Schreibe eine rekursive Funktion biszu(k), die eine Liste von Zahlen von 1 bis zu k ausgibt.

```
print biszu(6)
> [1, 2, 3, 4, 5, 6]
```

Rekursion mit Listen! (Warm-up)

Schreibe eine rekursive Funktion biszu(k), die eine Liste von Zahlen von 1 bis zu k ausgibt.

```
def biszu(k):
  if k == 1:
    return [1]
  else:
    return biszu(k - 1) + [k]
```

Rekursion mit Listen: Aufgaben

L1) Schreibe eine rekursive Funktion woerter(n), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen.

```
print woerter(3)
> ['AAA', 'AAB', 'ABA', 'ABB', 'BAA', 'BAB', 'BBA', 'BBB']
```

Rekursion mit Listen -- Hilfsblatt

Liste aus einer anderen Liste bauen:

```
a = [1, 13, 4]
b = []
for x in a:
  b.append(x + 1)
```

Oder nur:

```
b = [x + 1 for x in a]
```

```
c = []
for x in a:
  if x > 10:
    b.append(x * x)
```

Oder nur:

```
c = [x * x for x in a if x > 10]
```

Rekursion 3 -- Lösungen

Rekursion mit Listen: Aufgaben

L1) Schreibe eine rekursive Funktion woerter(n), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen.

```
print woerter(3)
> ['AAA', 'AAB', 'ABA', 'ABB', 'BAA', 'BAB', 'BBA', 'BBB']
```

Rekursion mit Listen: Aufgaben

L1) Schreibe eine rekursive Funktion woerter(n), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen.

```
woerter(3)
= ['AAA', 'AAB', 'ABA', 'ABB',
   'BAA', 'BAB', 'BBA', 'BBB']
```

= `'A'` + beliebiges Wort der Länge 2 aus A und B
oder `'B'` + beliebiges Wort der Länge 2 aus A und B

Rekursion mit Listen: Aufgaben

L1) Schreibe eine rekursive Funktion woerter(n), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen.

```
woerter(3)
= ['AAA', 'AAB', 'ABA', 'ABB',
   'BAA', 'BAB', 'BBA', 'BBB']
```

= 'A' + beliebiges Wort der Länge 2 aus A und B
oder 'B' + beliebiges Wort der Länge 2 aus A und B

= ['A' + w für alle w der Lange 2 aus A und B]
+ ['B' + w für alle w der Lange 2 aus A und B]

```
= ['A' + w for w in woerter(2)]
  + ['B' + w for w in woerter(2)]
```

Rekursion mit Listen: Aufgaben

L1) Schreibe eine rekursive Funktion woerter(n), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen.

```
woerter(3) = ['A' + w for w in woerter(2)] + ['B' + w for w in woerter(2)]

woerter(k) = ['A' + w for w in woerter(k - 1)]
           + ['B' + w for w in woerter(k - 1)]
```

Rekursion mit Listen: Aufgaben

L1) Schreibe eine rekursive Funktion woerter(n), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen.

```
woerter(3) = ['A' + w for w in woerter(2)] + ['B' + w for w in woerter(2)]

woerter(k) = ['A' + w for w in woerter(k - 1)]
           + ['B' + w for w in woerter(k - 1)]
```

Aber `woerter(1) = ['A' + 'B']`

Oder `woerter(0) = ['']` woher

```
woerter(1) = ['A' + w for w in woerter(0)] + ['B' + w for w in woerter(0)]
           = ['A' + w for w in ['']] + ['B' + w for w in ['']]
           = ['A' + ''] + ['B' + '']
           = ['A'] + ['B']
           = ['A', 'B']
```

funktioniert :-)

Rekursion mit Listen: Aufgaben

L1) Schreibe eine rekursive Funktion woerter(n), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen.

```
def woerter(k):
  if k == 0:
    return ['']
  else:
    return ['A' + w for w in woerter(k - 1)] + ['B' + w for w in woerter(k - 1)]
```

Rekursion mit Listen: Aufgaben

L1) Schreibe eine rekursive Funktion woerter(n), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen.

```
def woerter(k):
  if k == 0:
    return ['']
  else:
    res = []
    for w in woerter(k - 1):
      res.append('A' + w)
    for w in woerter(k - 1):
      res.append('B' + w)
    return res
```

```
res = ['A' + w for w in woerter(k - 1)]
      + ['B' + w for w in woerter(k - 1)]
```

Rekursion mit Listen: Aufgaben

L2) Schreibe eine rekursive Funktion gute_woerter(n), die alle Wörter der Länge n ausgibt, die nur aus A und B bestehen, und keine BB enthalten.

```
gute_woerter(3)
= ['AAA', 'AAB', 'ABA',
   'BAA', 'BAB']
```

= `'A'` + beliebiges gutes Wort der Länge 2 aus A und B
oder `'B'` + beliebiges gutes Wort der Länge 2 aus A und B, **das mit A beginnt**

= `['A' + w` für alle gute Wörter `w` der Lange 2 aus A und B`]`
`+ ['BA' + w` für alle gute Wörter `w` der Lange 1 aus A und B`]`

```
= ['A' + w for w in gute_woerter(2)]
  + ['BA' + w for w in gute_woerter(1)]
```

Rekursion mit Listen: Aufgaben

L2) Schreibe eine rekursive Funktion gute_woerter(n), die alle Wörter der Länge n ausgibt, die nur aus A und B bestehen, und keine BB enthalten.

```
def gute_woerter(k):
  if k == 0:
    return ['']
  elif k == 1:
    return ['A', 'B']
  else:
    return (['A' + w for w in gute_woerter(k - 1)] +
            ['BA' + w for w in gute_woerter(k - 2)])
```

Rekursion mit Listen: Aufgaben

L2) Schreibe eine rekursive Funktion gute_woerter(n), die alle Wörter der Länge n ausgibt, die nur aus A und B bestehen, und keine BB enthalten.

```
def gute_woerter(k):
  if k == 0:
    return ['']
  elif k == 1:
    return ['A', 'B']
  else:
    return (['A' + w for w in gute_woerter(k - 1)] +
            ['B' + w for w in gute_woerter(k - 1)  if w[0] != 'B'])
```

Rekursion mit Listen: Aufgaben

L3) Schreibe eine rekursive Funktion schoene_woerter(n), die alle Wörter der Länge n ausgibt, die nur aus A und B bestehen, und nicht mehr als zwei gleiche aufeinanderfolgende Buchstaben enthalten.

```
def schoene_woerter(k):
  if k == 0:
    return ['']
  else:
    return (['A' + w for w in schoene_woerter(k - 1)  if ???] +
            ['B' + w for w in schoene_woerter(k - 1)  if ???])
```

Rekursion mit Listen: Aufgaben

L3) Schreibe eine rekursive Funktion schoene_woerter(n), die alle Wörter der Länge n ausgibt, die nur aus A und B bestehen, und nicht mehr als zwei gleiche aufeinanderfolgende Buchstaben enthalten.

```
def schoene_woerter(k):
  if k == 0:
    return ['']
  else:
    return (['A' + w for w in schoene_woerter(k - 1)  if w[0:2] != 'AA'] +
            ['B' + w for w in schoene_woerter(k - 1)  if w[0:2] != 'BB'])
```

Rekursion mit Listen: Aufgaben

L4) Schreibe eine rekursive Funktion woerter(n, k), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen, und genau k-Mal ‘B’ enthalten.

```
def woerter(n, k):
  if n == 0:
    return ['']
  else:
    return (['A' + w for w in  ???] +
            ['B' + w for w in  ???])
```

Rekursion mit Listen: Aufgaben

L4) Schreibe eine rekursive Funktion woerter(n, k), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen, und genau k-Mal ‘B’ enthalten.

```
woerter(5, 3)
= ['A**BBB**', 'A**BABB**', 'A**BBAB**', 'A**BBBA**',
   'B**AABB**', 'B**ABAB**', 'B**ABBA**', 'B**BAAB**', 'B**BABA**', 'B**BBAA**']
= 'A' + Wort der Länge 4, davon 3 Bs, oder
  'B' + Wort der Länge 4, davon nur 2 Bs
```

Rekursion mit Listen: Aufgaben

L4) Schreibe eine rekursive Funktion woerter(n, k), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen, und genau k-Mal ‘B’ enthalten.

```
def woerter(n, k):
  if n == 0:
    return ['']
  else:
    return (['A' + w for w in woerter(n - 1, k)] +
            ['B' + w for w in woerter(n - 1, k - 1)])
```

Rekursion mit Listen: Aufgaben

L4) Schreibe eine rekursive Funktion woerter(n, k), die alle Wörter der Länge n ausgibt, die nur aus den Buchstaben A und B bestehen, und genau k-Mal ‘B’ enthalten.

```
def woerter(n, k):
  if n == 0 and k == 0:
    return ['']
  elif n == 0 and k != 0:
    return []
  else:
    return (['A' + w for w in woerter(n - 1, k)] +
            ['B' + w for w in woerter(n - 1, k - 1)])
```

Rekursion mit Listen: Aufgaben

L5) Schreibe eine rekursive Funktion untermengen(liste), die alle Untermengen der Liste liste ausgibt.

```
untermengen([1, 2, 3])

= [[1], [1, 2], [1, 3], [1, 2, 3],
   [], [2], [3], [2, 3]]
```

= `1 +` beliebige untermenge von `[2, 3]`
oder nur beliebige untermenge von `[2, 3]`

Rekursion mit Listen: Aufgaben

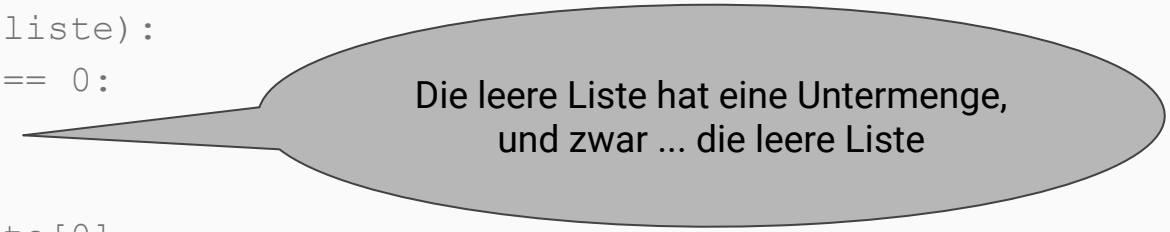
L5) Schreibe eine rekursive Funktion untermengen(liste), die alle Untermengen der Liste liste ausgibt.

```
def untermengen(liste):
  if len(liste) == 0:
    return []
  else:
    erstes = liste[0]
    rest = liste[1:]
    return [[erstes] + u for u in untermengen(rest)] + untermengen(rest)
```

Rekursion mit Listen: Aufgaben

L5) Schreibe eine rekursive Funktion untermengen(liste), die alle Untermengen der Liste liste ausgibt.

```
def untermengen(liste):
  if len(liste) == 0:
    return [[]]
  else:
    erstes = liste[0]
    rest = liste[1:]
    return [[erstes] + u for u in untermengen(rest)] + untermengen(rest)
```



Rekursion mit Listen: Aufgaben

L6) Schreibe eine rekursive Funktion untermengen(liste, k), die alle Untermengen der Liste liste ausgibt, **die genau k Elemente haben**.

```
def untermengen(liste, k):
  ???
```

Rekursion mit Listen: Aufgaben

L6) Schreibe eine rekursive Funktion untermengen(liste, k), die alle Untermengen der Liste liste ausgibt, **die genau k Elemente haben**.

```
def untermengen(liste, k):
  if len(liste) == 0:
    ???
  else:
    erstes = liste[0]
    rest = liste[1:]
    return ???
```

Rekursion mit Listen: Aufgaben

L6) Schreibe eine rekursive Funktion untermengen(liste, k), die alle Untermengen der Liste liste ausgibt, **die genau k Elemente haben**.

```
def untermengen(liste, k):
  if len(liste) == 0:
    if k == 0: return [[]]
    else: return []
  else:
    erstes = liste[0]
    rest = liste[1:]
    return ([[erstes] + u for u in untermengen(rest, k - 1)]
            + untermengen(rest, k))
```

Rekursion mit Listen: Aufgaben

L6) Schreibe eine rekursive Funktion untermengen(liste, k), die alle Untermengen der Liste liste ausgibt, **die genau k Elemente haben**.

```
def untermengen(liste, k):
  if len(liste) < k:
    return []
  if k == 0:
    return [[]]
  else:
    erstes = liste[0]
    rest = liste[1:]
    return ([[erstes] + u for u in untermengen(rest, k - 1)]
            + untermengen(rest, k))
```

Rekursion mit Listen: Aufgaben

L7) Schreibe eine rekursive Funktion ordnungen(liste), die alle möglichen Ordnungen der Liste liste ausgibt.

```
ordnungen([1, 2, 3])
= [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

= 1 am Anfang + beliebige Ordnung aller anderen Elementen, oder
  2 am Anfang + beliebige Ordnung aller anderen Elementen, oder
  3 am Anfang + beliebige Ordnung aller anderen Elementen

Rekursion mit Listen: Aufgaben

L7) Schreibe eine rekursive Funktion ordnungen(liste), die alle möglichen Ordnungen der Liste liste ausgibt.

```
def ordnungen(liste):
  if len(liste) <= 1:
    return [liste]
  else:
    res = []
    for am_anfang in liste:
      alle_andere = [x for x in liste if x != am_anfang]
      res = res + [[am_anfang] + o for o in ordnungen(alle_andere)]
    return res
```

Erinnerung: Die zwei optimalen Ferien

Nächstes Jahr hat Dennis mehr Freizeit und will sogar **zweimal** 5 Tage Skiurlaub machen.

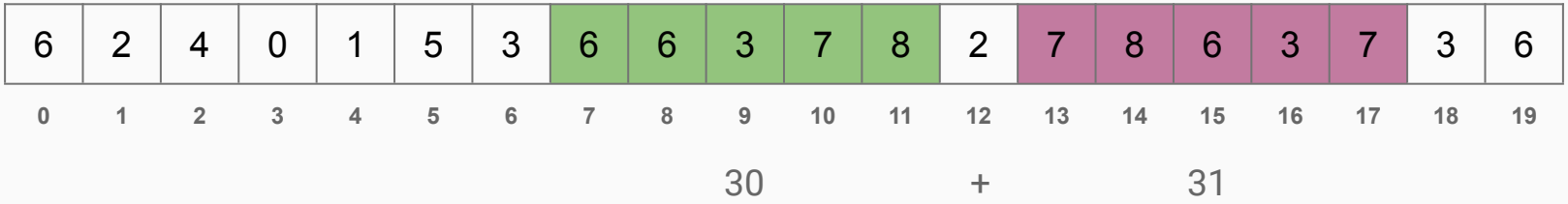
Findet für Dennis die optimalen Ferientermine (das heisst, zwei Abschnitte von 5 **aufeinanderfolgenden** Tagen mit maximaler Anzahl Sonnenstunden insgesamt).

6	2	4	0	1	5	3	6	6	3	7	8	2	7	8	6	3	7	3	6
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Die zwei optimalen Ferien

Nächstes Jahr hat Dennis mehr Freizeit und will sogar **zweimal** 5 Tage Skiurlaub machen.

Findet für Dennis die optimalen Ferientermine (das heisst, zwei Abschnitte von 5 **aufeinanderfolgenden** Tagen mit maximaler Anzahl Sonnenstunden insgesamt).



Die zwei optimalen Ferien

Nächstes Jahr hat Dennis mehr Freizeit und will sogar **k-Mal** 5 Tage Skiurlaub machen.

Findet für Dennis die optimalen Ferientermine (das heisst, **k** Abschnitte von 5 **aufeinanderfolgenden** Tagen mit maximaler Anzahl Sonnenstunden insgesamt).

6	2	4	0	1	5	3	6	6	3	7	8	2	7	8	6	3	7	3	6
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

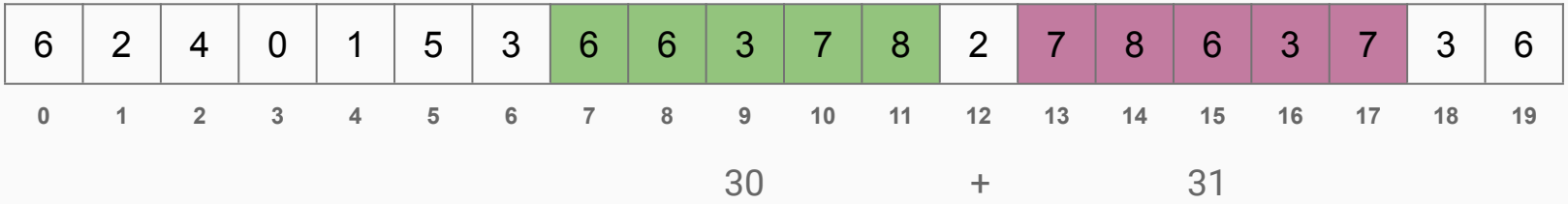
30 + 31

Die optimalen Ferien: Aufgabe 2

Finde die maximale Anzahl Sonnenstunden in **k** optimalen Ferienterminen.

```
def beste_urlaube(sonnenstunden, k):
    ...

print beste_urlaube([6,2,4,0,1,5,3,6,6,3,7,8,2,7,8,6,3,7,3,6], 2)
> 61
```

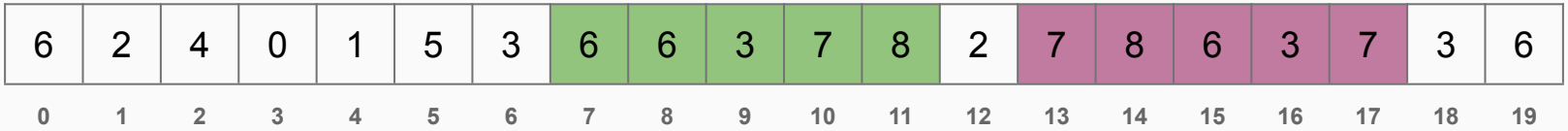


Die optimalen Ferien: Aufgabe 2

Finde die maximale Anzahl Sonnenstunden in **k** optimalen Ferienterminen.

```
def beste_urlaube(sonnenstunden, k):
```

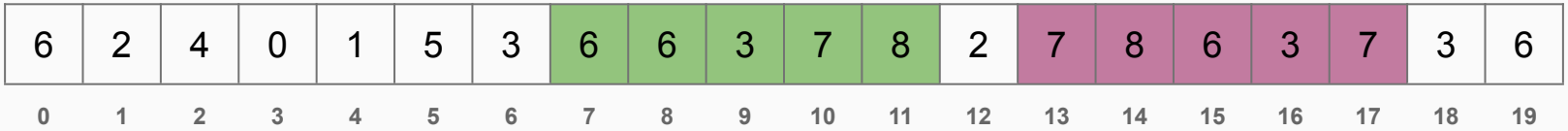
Entweder beginne ich der ersten Urlaub am tag 0,
oder nicht.
Ich nehme die bessere Möglichkeit von der zwei.



Die optimalen Ferien: Aufgabe 2

Schreibe eine Funktion, die die **k** optimalen Ferientermine berechnet.

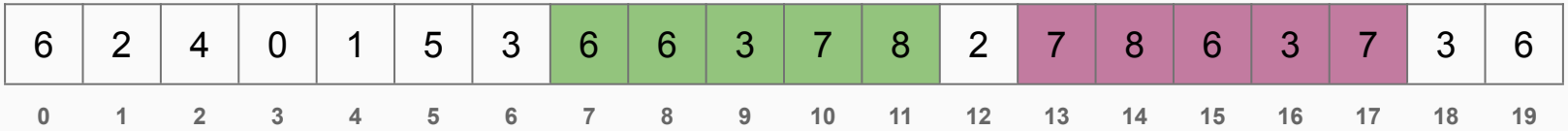
```
def beste_urlaube(sonnenstunden, k):
  vom_tag_0 = sum(sonnenstunden[0:5])
  return max(vom_tag_0 + beste_urlaube(sonnenstunden[5:], k - 1),
             beste_urlaube(sonnenstunden[1:], k))
```



Die optimalen Ferien: Aufgabe 2

Schreibe eine Funktion, die die **k** optimalen Ferientermine berechnet.

```
def beste_urlaube(sonnenstunden, k):
  if k == 0:
    return 0
  elif len(sonnenstunden) < 5:
    return -inf
  else:
    vom_tag_0 = sum(sonnenstunden[0:5])
    return max(vom_tag_0 + beste_urlaube(sonnenstunden[5:], k - 1),
               beste_urlaube(sonnenstunden[1:], k))
```

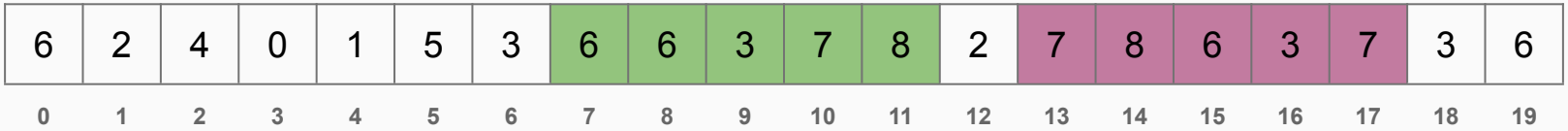


Die optimalen Ferien: Aufgabe 2

Schreibe eine Funktion, die die **k** optimalen Ferientermine berechnet.

```
def beste_urlaube(sonnenstunden, k):
  if k == 0:
    return 0
  elif len(sonnenstunden) < 5:
    return -inf
  else:
    vom_tag_0 = sum(sonnenstunden[0:5])
    return max(vom_tag_0 + beste_urlaube(sonnenstunden[5:], k - 1),
               beste_urlaube(sonnenstunden[1:], k))
```

Langsam weil rekursiv,
aber sehr einfach
zu programmieren!



Damen

B1) Schreibe eine Funktion damen(n), die eine Konfiguration von n Damen auf einem (n x n)-Schachbrett ausgibt, wobei sich keine zwei der Damen schlagen können.

```
> damen(8)
.......D
...D....
D.......
..D.....
.....D..
.D......
......D.
....D...
```

Damen

B1) Schreibe eine Funktion damen(n, **rows**), die die Reihen **rows zu einer Konfiguration** von n Damen auf einem (n x n)-Schachbrett **vervollständigt**, in der sich keine zwei der Damen schlagen können.

```
> damen(8, [.......D
            ...D....
            D.......
            ..D.....])
.......D
...D....
D.......
..D.....
.....D..
.D......
......D.
....D...
```

Damen

B1) Schreibe eine Funktion damen(n, **rows**), die die Reihen **rows zu einer Konfiguration** von n Damen auf einem (n x n)-Schachbrett **vervollständigt**, in der sich keine zwei der Damen schlagen können.

`def damen(8, rows):`

Für alle mögliche nächsten Reihen `row`,

versuche `rows + [row]` mithilfe `damen(8, rows + [row])` zu vervollständigen, und gib die Lösung zurück, wenn eine gefunden wird.

Sonst geht es nicht.

Damen

B1) Schreibe eine Funktion damen(n, **rows**), die die Reihen **rows zu einer Konfiguration** von n Damen auf einem (n x n)-Schachbrett **vervollständigt**, in der sich keine zwei der Damen schlagen können.

```
def damen(8, rows):
    Für alle Spalten col,
        wenn ich in der nächsten Reihe eine Dame in die Spalte col platzieren kann,
        ohne dass sie sich mit den vorherigen Damen schlägt,
            erzeuge solche Reihe row mit Deme in der Spalte col, und
            versuche rows + [row]  mithilfe damen(8, rows + [row])  zu vervollständigen,
            und gib die Lösung zurück, wenn eine gefunden wird.
    Sonst geht es nicht.
```