Aufgabe 2.1

```
A = {{0, 4, 3}, {0, 7, 5}, {0, 3, 7}}
```

```
{{0, 4, 3}, {0, 7, 5}, {0, 3, 7}}
```

```
b = {8, 14, 6}
```

```
{8, 14, 6}
```

Bestimme eine partikuläre Lösung für A`x==b`:

```
LinearSolve[A, b]
```

```
{0, 2, 0}
```

Bestimme Rang und Kern der Matrix:

```
MatrixRank[A]
```

```
2
```

```
k = NullSpace[A]
```

```
{{1, 0, 0}}
```

Aufgabe 2.2

a) Der Parameter `k` bestimmt die obere Grenze der Iteration:

```
a[k_] := Table[2^n, {n, k}]
```

```
a[8]
```

```
{2, 4, 8, 16, 32, 64, 128, 256}
```

b)

```
liste[a_, b_, n_] := Range[a, b, (b-a)/(n-1)]
```

```
liste[1, 11, 6]
```

```
{1, 3, 5, 7, 9, 11}
```

c)

```
mat [n_, m_ ] := Table[ij, {i, n}, {j, m }]
```

```
mat [3, 4] // MatrixForm
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \end{pmatrix}$$

d) Es gibt zwar die Funktion `Binomial[n,k]`, aber auch die kann man selbst definieren:

```
bk[_, 0] = 1;
bk[n_, n_] = 1;
bk[n_, k_] := bk[n-1, k-1] + bk[n-1, k]
```

Damit kann man dann das Pascalsche Dreieck erzeugen: `k` iteriert in jeder Zeile von 0 bis zur Zeilennum-

mer `i`; `i` läuft von 0 bis zum Eingabewert `n`.

```
pascal[n_] := Table[bk[i, k], {i, 0, n}, {k, 0, i}]
```

```
pascal[5] // TableForm
```

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
```

e) Der Übersichtlichkeit halber kann man eine Hilfsfunktion definieren, die eine gegebene Liste gemäß der Fibonacci-Regel um ein Element erweitert:

```
fibgrow[L_List /; Length[L] > 1] := Append[L, L[[-1]] + L[[-2]]];
```

Damit dann `fib[n]`, um die Liste zu erzeugen:

```
fib[1] = {1};
```

```
fib[2] = {1, 1};
```

```
fib[n_Integer /; n > 2] := fibgrow[fib[n - 1]]
```

```
fib[10]
```

```
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55}
```

f) Die Anzahl der Zeilen einer Matrix `M` ist `Length[M]`. Wenn man voraussetzt, dass `M` rechteckig ist, also alle Zeilen gleich lang sind, entspricht die Länge der ersten Zeile (`Length[M[[1]]]`) der Anzahl der Spalten.

```
colsum [M_] := Table[Sum_{i=1}^{Length[M]} M[[i, j]], {j, Length[M[[1]]]}]
```

```
colsum [{{1, 2}, {3, 4}}]
```

```
{4, 6}
```

Aufgabe 2.3

```
h[x_ /; x < 0] = 0;
```

```
h[_] = 1;
```

Aufgabe 2.4

Das Problem besteht darin, den Tausch ohne lokale Variable zu vollführen, also ohne Werte zwischenzuspeichern. Idee: Eine Hilfsfunktion `swapcol` gibt ein einzelnes Listenelement für den Index `i` zurück, vertauscht dabei aber die Elemente an den Indizes `c1` und `c2` miteinander.

```
swapcol[v_, c1_, c2_, i_] /; i == c1 := v[[c2]];
swapcol[v_, c1_, c2_, i_] /; i == c2 := v[[c1]];
swapcol[v_, c1_, c2_, i_] := v[[i]];
```

```
swap[v_, c1_, c2_] := Table[swapcol[v, c1, c2, i], {i, 1, Length[v]}]
```

```
swap[{1, 2, 3, 4, 5}, 2, 4]
```

```
{1, 4, 3, 2, 5}
```

Aufgabe 2.5

Wenn die Liste bereits sortiert ist, wird sie unverändert zurückgegeben.

```
sort[v_List?OrderedQ] := v
```

Andernfalls muss es mindestens zwei Elemente geben, die nicht in der erwünschten Reihenfolge nebeneinander stehen, die folglich vertauscht werden, anschließend wird rekursiv weiter sortiert. Das Muster mit drei *Blanks* trifft dabei auf null oder beliebig viele Elemente.

```
sort[{a___, i_, j_, b___} /; i > j] := sort[{a, j, i, b}]
```