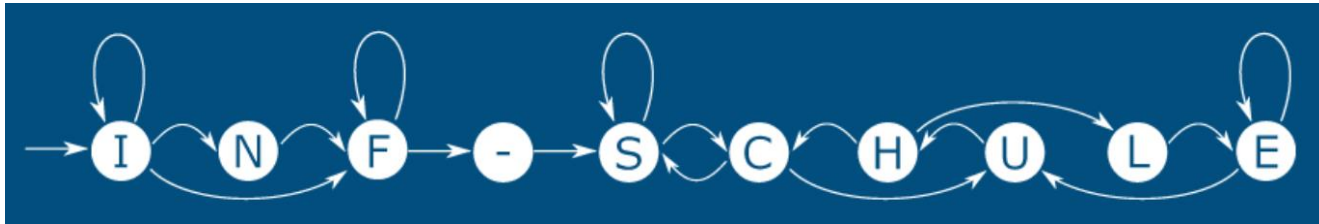


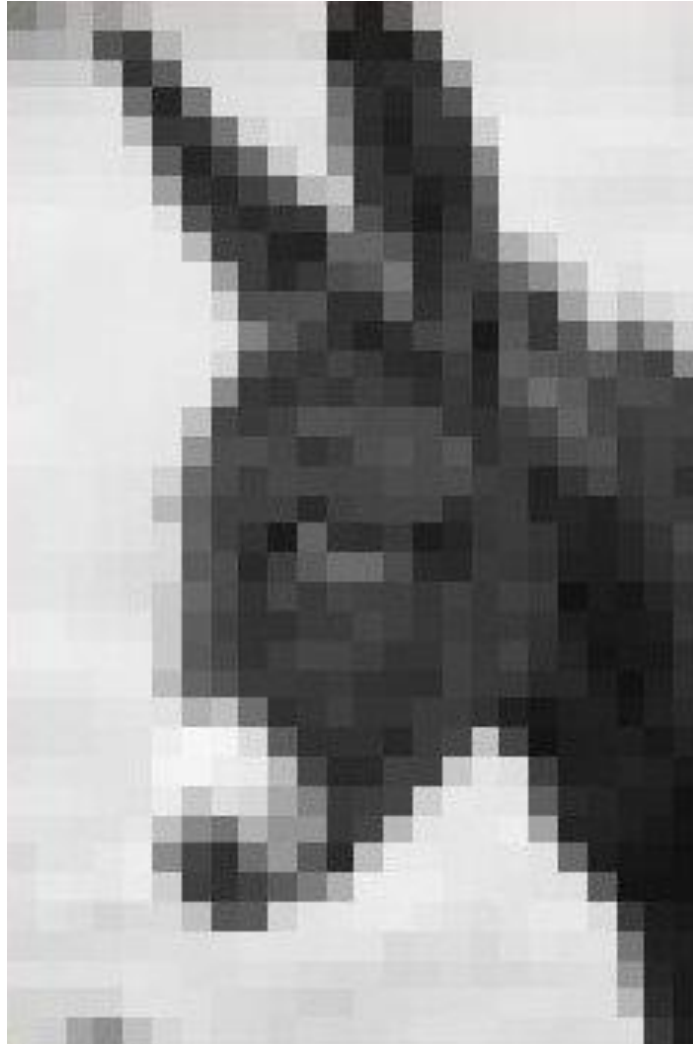
# Datenstrukturen II

## - *PGM* -



Samuel Dietz

2023



Ziel ist es, **Datenstrukturen zur Verwaltung komplexer Daten** einzuführen. Dabei steht das Listenkonzept im Vordergrund.

Wir werden das hier im Kontext von Miniprojekten durchführen. Dabei steigt der Komplexitätsgrad innerhalb der Projekte und auch in der Abfolge der Projekte. Im Unterricht reicht es, **sich auf Teile zu beschränken**.

*Die Bearbeitung der Projekte setzt einen sicheren Umgang mit Kontrollstrukturen und dem Funktionskonzept sowie einer Python-Programmierungsumgebung voraus.*

**Zeitansatz im Weiterbildungskurs:** 4\*90 Min.

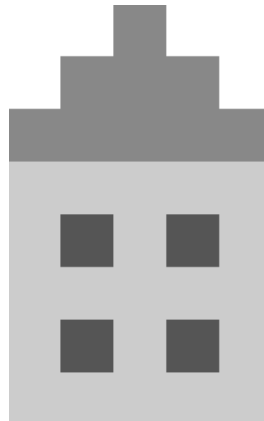
**Zeitansatz im Unterricht:** hängt sehr von den behandelten Projekten ab; mindestens 6 Stunden

PGM

# Portable Graymap (PGM)

**PGM** steht für Portable Graymap. PGM ist ein Format zur Darstellung von Bildern, die nur aus Graustufen bestehen.

```
P2
# haus.pgm
5 8
15
15 15 8 15 15
15 8 8 8 15
8 8 8 8 8
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
```



Die erste Zeile "P2" kennzeichnet das PGM-Format in der ASCII-Version.

Die zweite Zeile enthält einen Kommentar, den man auch weglassen kann.

Die beiden Zahlen "5 8" in der dritten Zeile beschreiben die Aufteilung der **Pixel in Spalten und Zeilen** (hier 5 Spalten und 8 Zeilen).

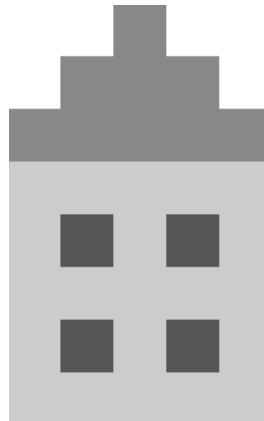
Die Zahl "15" in der vierten Zeile beschreibt die Anzahl der Graustufen - hier von Stufe 0 bis Stufe 15.

In den folgenden Zeilen werden die einzelnen Grauwerte mit Hilfe von Zahlen dargestellt. Diese sind hier - der besseren Lesbarkeit wegen - bereits im Quelltext genauso angeordnet wie in der beabsichtigten Bilddarstellung. Eine solche Anordnung ist aber nicht erforderlich. Man könnte z.B. alle Zahlen in eine einzige Zeile schreiben oder auch jede dieser Zahlen in eine neue Zeile.

## Problem:

Wie verwaltet man die Daten eines PGM-Quelltextes?

```
P2
# haus.pgm
5 8
15
15 15 8 15 15
15 8 8 8 15
8 8 8 8 8
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
```



Zeichenkette (über mehrere Zeilen)

```
quelltext = """
P2
# haus.pgm
5 8
15
15 15 8 15 15
15 8 8 8 15
8 8 8 8 8
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
"""

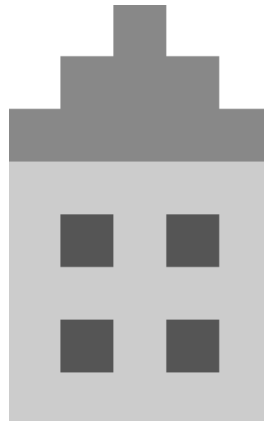
print(quelltext)
print(quelltext[4])
print(quelltext[80])
```

**Problem:**

Wie verwaltet man die Daten eines PGM-Quelltextes?

**Strukturierte Daten mit Variablen**

```
P2
# haus.pgm
5 8
15
15 15 8 15 15
15 8 8 8 15
8 8 8 8 8
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
```



```
kennzeichnung = "P2"
kommentar = "# haus.pgm"
spalten = 5
zeilen = 8
graustufen = 15
bilddaten = [15, 15, 8, 15, 15,
              15, 8, 8, 8, 15,
              8, 8, 8, 8, 8,
              12, 12, 12, 12, 12,
              12, 5, 12, 5, 12,
              12, 12, 12, 12, 12,
              12, 5, 12, 5, 12,
              12, 12, 12, 12, 12]
grafik = (kennzeichnung,
          kommentar, spalten, zeilen,
          graustufen, bilddaten)
```

# Datenverwaltung

Ein **Datentyp** beschreibt eine Menge von Datenobjekten, die alle die gleiche Struktur haben und mit denen die gleichen Operationen ausgeführt werden können.

**Datenstrukturen** ermöglichen es, strukturierte Daten als Einheit zu verwalten.

Datentyp: Zeichenkette

Datentyp: ganze Zahl

Datenstruktur: Liste

Datentyp: Liste

Datenstruktur: Tupel

Datentyp: Tupel

```
kennzeichnung = "P2"
kommentar = "# haus.pgm"
spalten = 5
zeilen = 8
graustufen = 15
bilddaten = [15, 15, 8, 15, 15,
              15, 8, 8, 8, 15,
              8, 8, 8, 8, 8,
              12, 12, 12, 12, 12,
              12, 5, 12, 5, 12,
              12, 12, 12, 12, 12,
              12, 5, 12, 5, 12,
              12, 12, 12, 12, 12]

grafik = (kennzeichnung,
          kommentar, spalten, zeilen,
          graustufen, bilddaten)
```



```
kennzeichnung = "P2"  
kommentar = "# haus.pgm"  
spalten = 5  
zeilen = 8  
graustufen = 15  
bilddaten = [15, 15, 8, 15, 15,  
              15, 8, 8, 8, 15,  
              8, 8, 8, 8, 8,  
              12, 12, 12, 12, 12,  
              12, 5, 12, 5, 12,  
              12, 12, 12, 12, 12,  
              12, 5, 12, 5, 12,  
              12, 12, 12, 12, 12]  
grafik = (kennzeichnung, kommentar, spalten, zeilen, graustufen,  
          bilddaten)
```

Programmfenster

```
>>> grafik  
('P2', '# haus.pgm', 5, 8, 15, [15, 15, 8, 15, 15, 15, 8, 8, 8, 15,  
8, 8, 8, 8, 8, 12, 12, 12, 12, 12, 12, 5, 12, 5, 12, 12, 12, 12, 12,  
12, 12, 5, 12, 5, 12, 12, 12, 12, 12, 12])
```

Ausführfenster

Grafik invertieren

**Zielsetzung:**

Ein PGM-Graustufenbild soll invertiert werden.



## Lösungsansatz:

Quelltext laden -> Quelltext als Zeichenkette -> in strukturierte Daten umwandeln -> ...



```

quelltext =
'''
P2
# haus.pgm
5 8
15
15 15 8 15 15
15 8 8 8 15
8 8 8 8 8
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
'''

```



```


grafik =
('P2',
'# haus.pgm',
5, 8,
15,
[15, 15, 8, 15, 15,
15, 8, 8, 8, 15,
8, 8, 8, 8, 8,
12, 12, 12, 12, 12,
12, 5, 12, 5, 12,
12, 12, 12, 12, 12,
12, 5, 12, 5, 12,
12, 12, 12, 12, 12])

```




## Lösungsansatz:


... -> strukturierte Bilddaten -> strukturierte Bilddaten zum invertierten Bild erzeugen -> ...



```
grafik =  
( 'P2',  
  '# haus.pgm',  
  5, 8,  
  15,  
  [15, 15, 8, 15, 15,  
   15, 8, 8, 8, 15,  
   8, 8, 8, 8, 8,  
   12, 12, 12, 12, 12,  
   12, 5, 12, 5, 12,  
   12, 12, 12, 12, 12,  
   12, 5, 12, 5, 12,  
   12, 12, 12, 12, 12] )
```




```
grafikNeu =  
( 'P2',  
  '# haus.pgm neu',  
  5, 8,  
  15,  
  [0, 0, 7, 0, 0,  
   0, 7, 7, 7, 0,  
   7, 7, 7, 7, 7,  
   3, 3, 3, 3, 3,  
   3, 10, 3, 10, 3,  
   3, 3, 3, 3, 3,  
   3, 10, 3, 10, 3,  
   3, 3, 3, 3, 3] )
```




## Lösungsansatz:


... -> strukturierte Daten -> in Quelltext umwandeln -> Quelltext speichern



```
grafikNeu =  
( 'P2',  
  '# haus.pgm neu',  
  5, 8,  
  15,  
  [0, 0, 7, 0, 0,  
    0, 7, 7, 7, 0,  
    7, 7, 7, 7, 7,  
    3, 3, 3, 3, 3,  
    3, 10, 3, 10, 3,  
    3, 3, 3, 3, 3,  
    3, 10, 3, 10, 3,  
    3, 3, 3, 3, 3])
```



```
quelltextNeu = '''  
P2  
# haus.pgm neu  
5 8  
15  
0 0 7 0 0 0 7 7 7 0  
7 7 7 7 7 3 3 3 3 3  
3 10 3 10 3 3 3 3 3  
3 3 10 3 10 3 3 3 3  
3 3  
'''
```

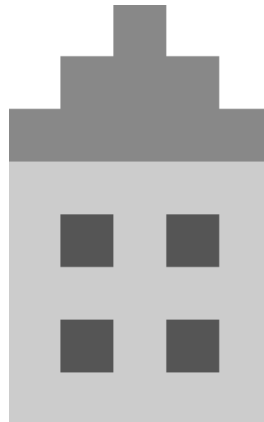


Bilddaten invertieren

# Verarbeitung von Listen

Ziel ist es, eine Verarbeitungseinheit zum Invertieren der Bilddaten zu entwickeln.

```
P2
# haus.pgm
5 8
15
15 15 8 15 15
15 8 8 8 15
8 8 8 8 8
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
12 5 12 5 12
12 12 12 12 12
```



```
P2
# hausneu.pgm
5 8
15
0 0 7 0 0
0 7 7 7 0
7 7 7 7 7
3 3 3 3 3
3 10 3 10 3
3 3 3 3 3
3 10 3 10 3
3 3 3 3 3
```

```
[15, 15, 8, 15, 15, 15, 8,
8, 8, 15, 8, 8, 8, 8, 8, 12,
12, 12, 12, 12, 12, 5, 12,
5, 12, 12, 12, 12, 12, 12,
12, 5, 12, 5, 12, 12, 12,
12, 12, 12]
```

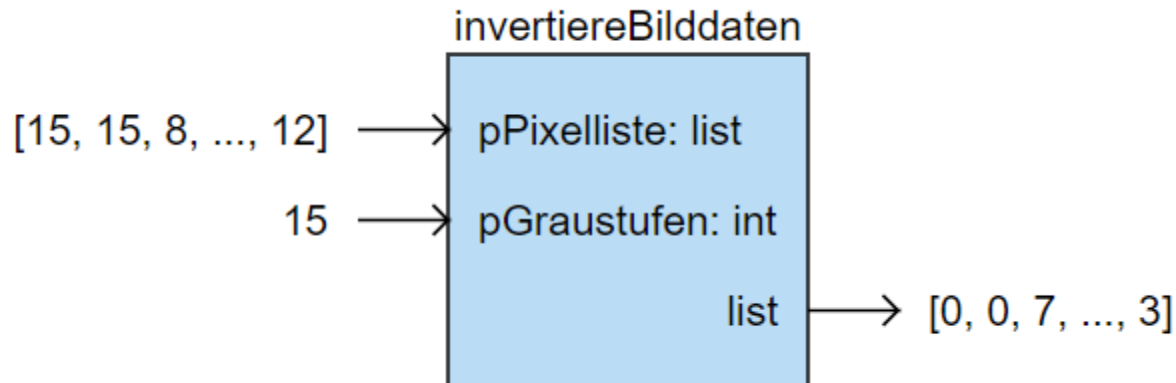


```
[0, 0, 7, 0, 0, 0, 7, 7, 7,
0, 7, 7, 7, 7, 7, 3, 3, 3,
3, 3, 3, 10, 3, 10, 3, 3, 3,
3, 3, 3, 3, 10, 3, 10, 3, 3,
3, 3, 3, 3]
```



# Verarbeitung von Listen

Ziel ist es, eine Verarbeitungseinheit zum Invertieren der Bilddaten zu entwickeln.



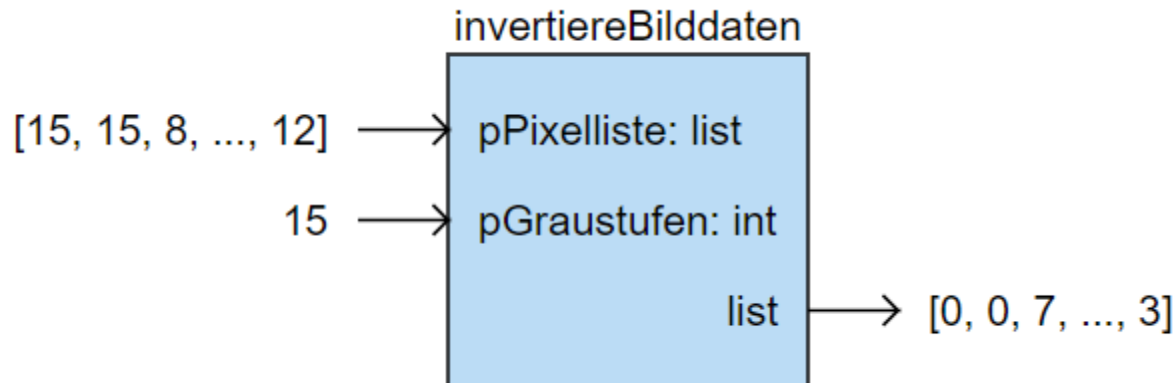
```
[15, 15, 8, 15, 15, 15, 8,
8, 8, 15, 8, 8, 8, 8, 8, 12,
12, 12, 12, 12, 12, 5, 12,
5, 12, 12, 12, 12, 12, 12,
12, 5, 12, 5, 12, 12, 12,
12, 12, 12]
```



```
[0, 0, 7, 0, 0, 0, 7, 7, 7,
0, 7, 7, 7, 7, 7, 3, 3, 3,
3, 3, 3, 10, 3, 10, 3, 3, 3,
3, 3, 3, 3, 10, 3, 10, 3, 3,
3, 3, 3, 3]
```

# Verarbeitung von Listen

Ziel ist es, eine Verarbeitungseinheit zum Invertieren der Bilddaten zu entwickeln.



## Aufgabe:

Implementiere die Funktion und teste die Implementierung.

Nutze die Datei „*F 18 invertieren2geruest.py*“.

**Aufgabe:**

Teste die folgende Implementierung der Funktion `invertiereBilddaten`. Was fällt auf?  
Nutze die Datei „*F 19 invertieren2seiteneffekt.py*“.

```
# Funktionsdefinition
def invertiereBilddaten(pBilddaten, pGraustufen):
    for i in range(len(pBilddaten)):
        pBilddaten[i] = pGraustufen - pBilddaten[i]
    return pBilddaten

# Funktionsaufruf
graustufen = 15
bilddaten = [15, 15, 8, 15, 15,
              15, 8, 8, 8, 15,
              8, 8, 8, 8, 8,
              12, 12, 12, 12, 12,
              12, 5, 12, 5, 12,
              12, 12, 12, 12, 12,
              12, 5, 12, 5, 12,
              12, 12, 12, 12, 12]

bilddatenNeu = invertiereBilddaten(bilddaten, graustufen)
print(bilddaten)
print(bilddatenNeu)
```

Wenn eine Funktion den Wert einer globalen Variablen verändert, liegt ein **Seiteneffekt** vor.

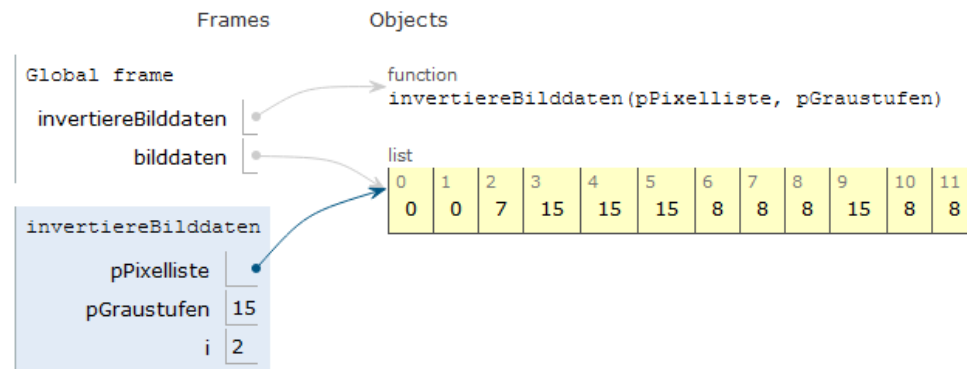
Python 3.6  
([known limitations](#))

```

1 def invertiereBilddaten(pPixelliste, pGraustufen):
2     for i in range(len(pPixelliste)):
3         pPixelliste[i] = pGraustufen - pPixelliste[i]
4     return pPixelliste
5
6
7 # Test
8
9 bilddaten = [15, 15, 8, 15, 15,
10             15, 8, 8, 8, 15,
11             8, 8, 8, 8, 8,
12             12, 12, 12, 12, 12,
13             12, 5, 12, 5, 12,
14             12, 12, 12, 12, 12,
15             12, 5, 12, 5, 12,
16             12, 12, 12, 12, 12]
17 bilddatenNeu = invertiereBilddaten(bilddaten, 15)

```

Print output (drag lower right corner to resize)



Seiteneffekte führen dazu, dass das Verhalten von Funktionen schwer zu durchschauen ist. Insbesondere bei komplexeren Programmen verliert man leicht den Überblick, welche (beabsichtigten und auch unbeabsichtigten) Nebenwirkungen ein Funktionsaufruf mit Seiteneffekten hat. Man versucht daher, Seiteneffekte möglichst zu vermeiden.

**Aufgabe:**

Teste auch diese Implementierung. Worin besteht der Unterschied zur vorherigen?

```
# Funktionsdefinition
def invertiereBilddaten(pBilddaten, pGraustufen):
    for i in range(len(pBilddaten)):
        pBilddaten[i] = pGraustufen - pBilddaten[i]
    return pBilddaten

# Funktionsaufruf
graustufen = 15
bilddaten = [15, 15, 8, 15, 15,
              15, 8, 8, 8, 15,
              8, 8, 8, 8, 8,
              12, 12, 12, 12, 12,
              12, 5, 12, 5, 12,
              12, 12, 12, 12, 12,
              12, 5, 12, 5, 12,
              12, 12, 12, 12, 12]

bilddatenNeu = invertiereBilddaten(bilddaten, graustufen)
print(bilddaten)
print(bilddatenNeu)
```

# Vermeidung von Seiteneffekten

Beachte: Hier wird ein neues Listenobjekt erzeugt.

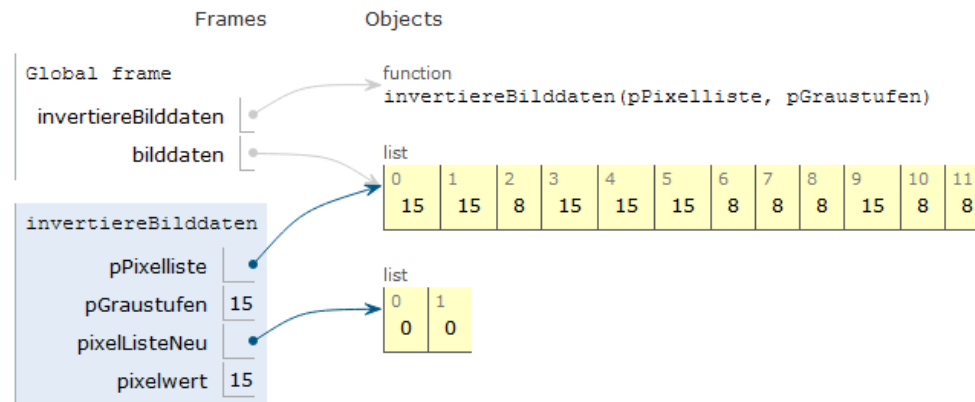
Python 3.6  
(known limitations)

```

1 def invertiereBilddaten(pPixelliste, pGraustufen):
2     pixelListeNeu = []
3     for pixelwert in pPixelliste:
4         pixelListeNeu = pixelListeNeu + [pGraustufen -
5     return pixelListeNeu
6
7 # Test
8
9 bilddaten = [15, 15, 8, 15, 15,
10             15, 8, 8, 8, 15,
11             8, 8, 8, 8, 8,
12             12, 12, 12, 12, 12,
13             12, 5, 12, 5, 12,
14             12, 12, 12, 12, 12,
15             12, 5, 12, 5, 12,
16             12, 12, 12, 12, 12]
17 bilddatenNeu = invertiereBilddaten(bilddaten, 15)

```

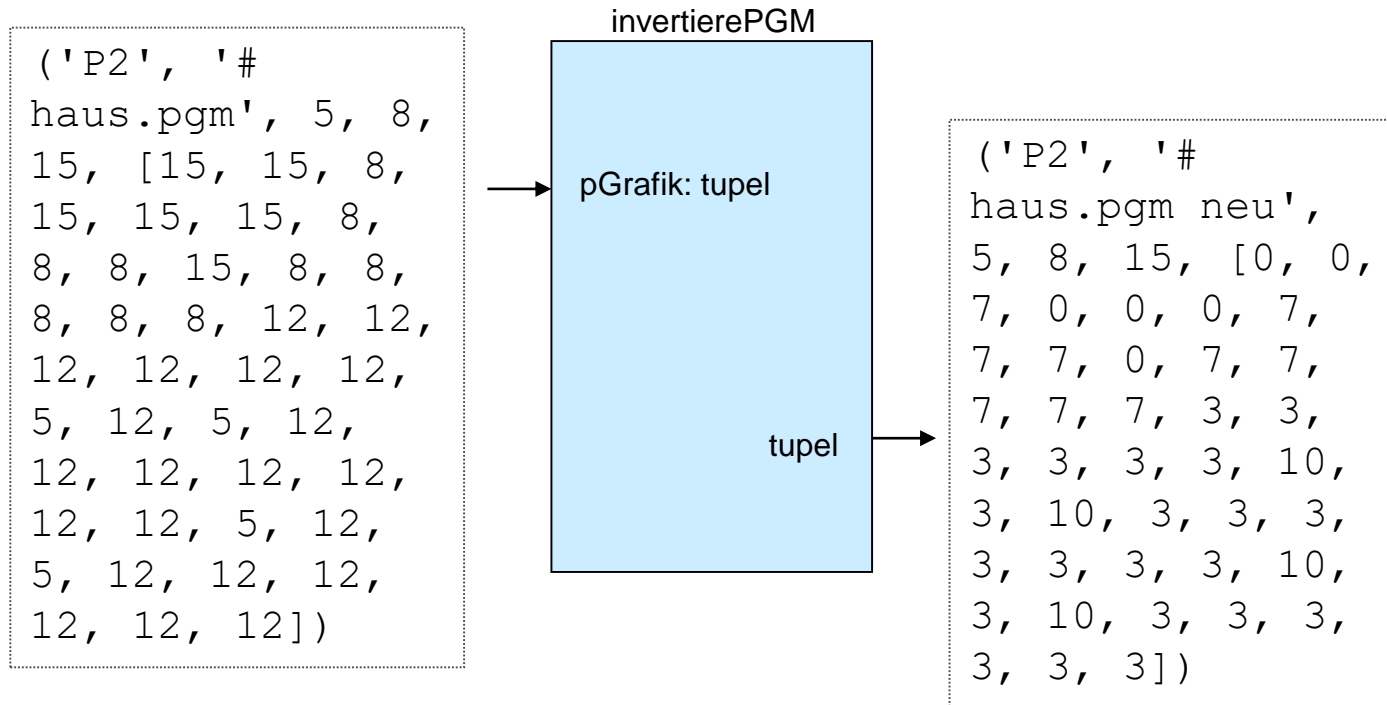
Print output (drag lower right corner to resize)



## Miniprojekt „Grafikdaten invertieren“

# Verarbeitung der gesamten Grafik

Ziel ist es, eine Verarbeitungseinheit zum Invertieren der Bilddaten zu entwickeln.





Eine **Tupel** ist eine Datenstruktur, bei der mehrere Daten zu einer Einheit (einem Datensatz) zusammengefasst werden.

```
datum = (24, 'Jul', 2022)
uhrzeit = (12, 44, 21)
zeitverschiebung = ('+', 2, 0)
zeitstempel = (datum, uhrzeit,
zeitverschiebung)
```

```
>>> zeitstempel
((24, 'Jul', 2022), (12, 44, 21),
('+', 2, 0))
```

Beispiel 1

```
kennzeichnung = "P2"
kommentar = "# haus.pgm"
spalten = 5
zeilen = 8
graustufen = 15
bilddaten = [15, 15, 8, 15, 15,
              15, 8, 8, 8, 15,
              8, 8, 8, 8, 8,
              12, 12, 12, 12, 12,
              12, 5, 12, 5, 12,
              12, 12, 12, 12, 12,
              12, 5, 12, 5, 12,
              12, 12, 12, 12, 12]
grafik = (kennzeichnung,
kommentar, spalten, zeilen,
graustufen, bilddaten)
```

Beispiel 2

# Verarbeitung von Tupel

Tupel weisen einige Ähnlichkeiten mit Listen auf, unterscheiden sich aber in einigen Punkten wesentlich von Listen.

Ein lesender Zugriff auf Tupelelemente erfolgt genau wie bei Listen. Ein schreibender Zugriff auf Tupelelemente ist - anders als bei Listen - nicht möglich.

```
>>> datum = (26, 'Jan', 2022)
>>> datum[0]
26
>>> datum[1]
'Jan'
>>> datum[2] = 2023
Traceback (most recent call
last):
  File "<pyshell>", line 1, in
<module>
TypeError: 'tuple' object does
not support item assignment
```

Tupel verarbeiten

```
>>> datum = [26, 'Jan', 2022]
>>> datum[0]
26
>>> datum[1]
'Jan'
>>> datum[2] = 2023
>>> datum
[26, 'Jan', 2023]
```

Liste verarbeiten

## Aufgabe:

Probiere das selbst aus.

# Verarbeitung von Tupel

Man kann ein Tupel nicht dadurch abändern, dass man ein Element durch ein anderes ersetzt. Wenn man ein Tupel abändern will, muss man ein neues Tupelobjekt erzeugen.

```
>>> datum = (26, 'Jan', 2022)
>>> datum[0]
26
>>> datum[1]
'Jan'
>>> datum[2] = 2023
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'tuple' object does not support
item assignment
```

Geht nicht: Element abändern

```
>>> datum = (26, 'Jan', 2022)
>>> datum = (datum[0], datum[1], 2023)
>>> datum
(26, 'Jan', 2023)
```

Neues Tupelobjekt erzeugen

## Aufgabe:

Probiere das selbst aus.

# Verarbeitung von Tupel

Tupel lassen sich auf einfache Weise auspacken und zusammenpacken. Auf diese Weise lässt sich ein leicht abgewandeltes Tupel erzeugen.

```
>>> tag = 26
>>> monat = 'Jan'
>>> jahr = 2022
>>> datum = (tag, monat, jahr)
>>> datum
(26, 'Jan', 2022)
```

```
>>> datum = (26, 'Jan', 2022)
>>> (tag, monat, jahr) = datum
>>> tag
26
>>> monat
'Jan'
>>> jahr
2022
```

Tupel zusammenpacken

```
>>> datum = (26, 'Jan', 2022)
>>> (tag, monat, jahr) = datum
>>> tag = tag + 1
>>> datumNeu = (tag, monat, jahr)
>>> datumNeu
(27, 'Jan', 2022)
```

Tupel abändern

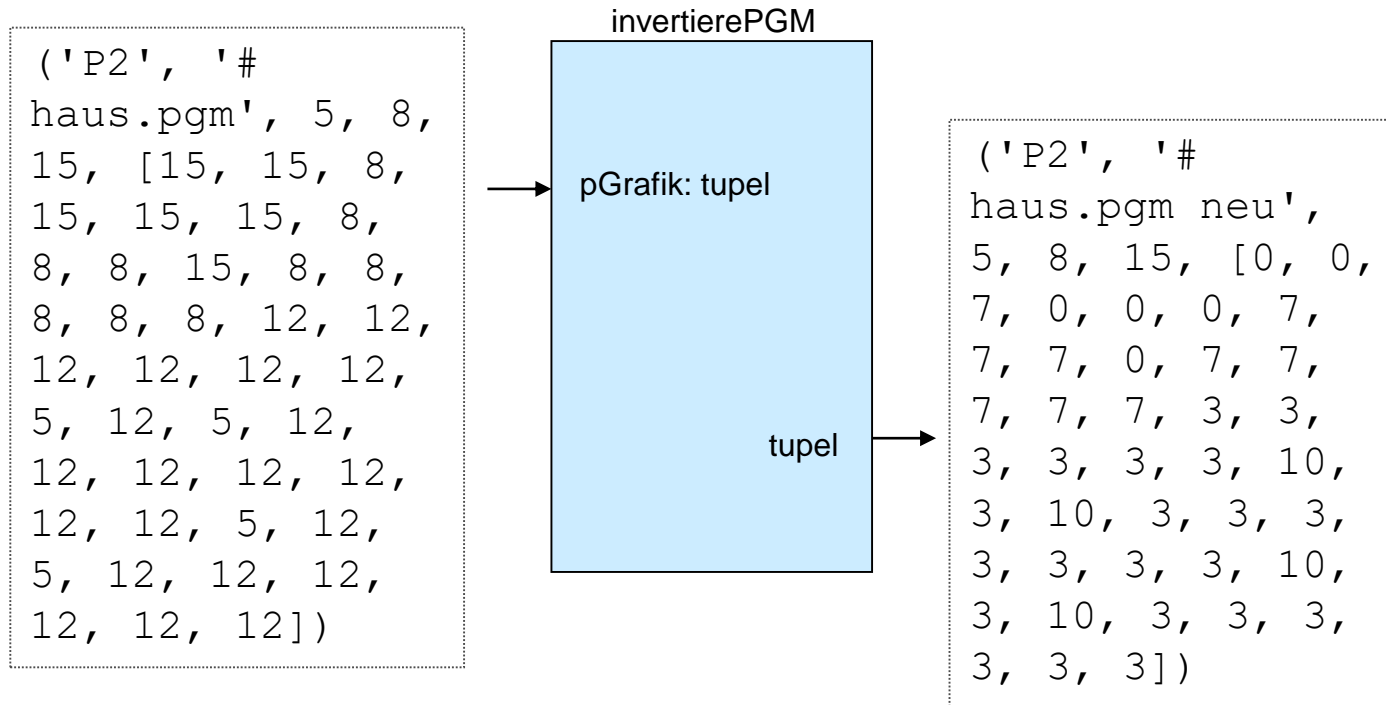
Tupel auspacken

## Aufgabe:

Probiere das selbst aus.

# Verarbeitung der gesamten Grafik

Ziel ist es, eine Verarbeitungseinheit zum Invertieren der Bilddaten zu entwickeln.



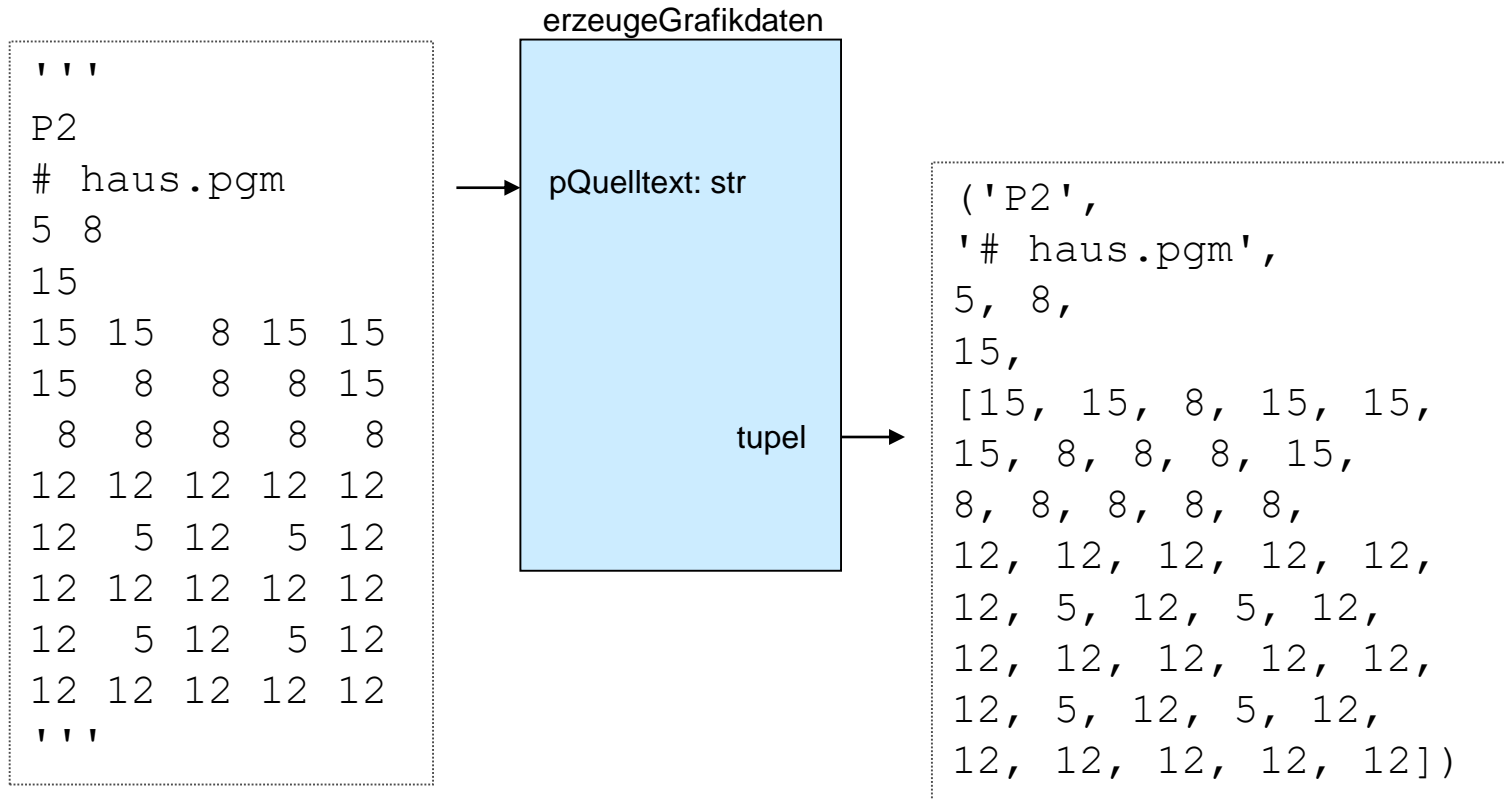
## Aufgabe:

Entwickle eine Funktion `invertierePGM` und teste die Implementierung. Benutze dabei die Verarbeitung von Tupeln.

## Umwandlung in Quelltexte

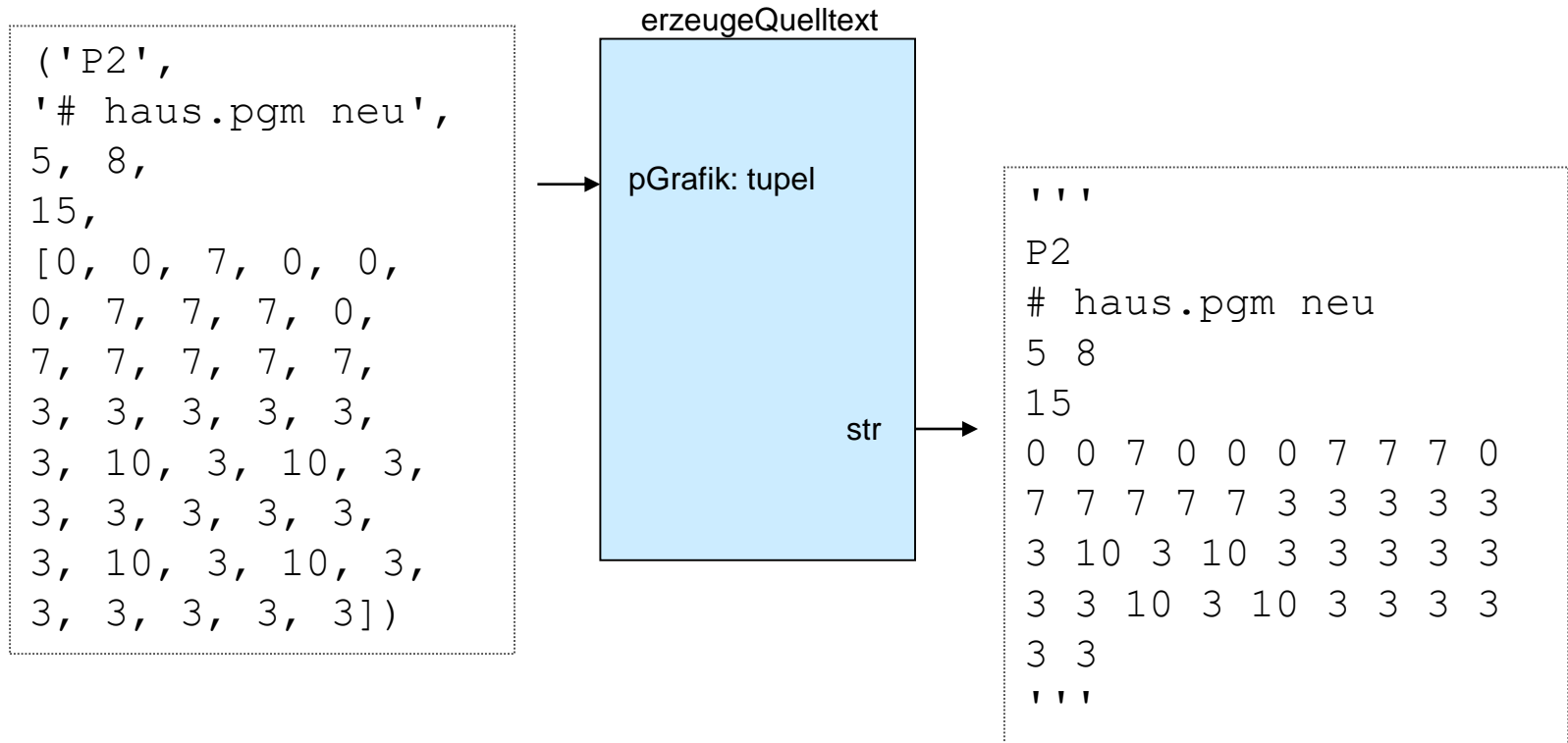
**Problem 1:**

Der Quelltext als Zeichenkette soll in strukturierte Daten umgewandelt werden.



**Problem 2:**

Strukturierte Daten sollen in einen Quelltext als Zeichenkette umgewandelt werden.





Problem 2 ist eher einfach.

```
def erzeugeQuelltext(pGrafik):  
    (kennzeichnung, kommentar, spalten, zeilen, grauwerte, bilddaten)  
= pGrafik  
    quelltext = '\n'  
    quelltext = quelltext + kennzeichnung  
    if kommentar != '':  
        quelltext = quelltext + '\n' + kommentar + '\n'  
    # ...  
    return quelltext
```

## Aufgabe:

Ergänze die fehlenden Teile und teste die Funktion.

Problem 1 ist viel schwieriger. Man muss die Zeichenkette zerlegen und Daten ggf. umwandeln..

```
'P2\n# haus.pgm\n5 8\n15\n15 15 8 15 15\n15 8 8 8 15 ...'
```

↑ ↑ ↑

0 2 3

↑ ↑ ↑

13 14  
15

```
kennzeichnung = quelltext[0:2]
kommentar = quelltext[3:13]
spalten = int(quelltext[14:15])
...
```

## Aufgabe:

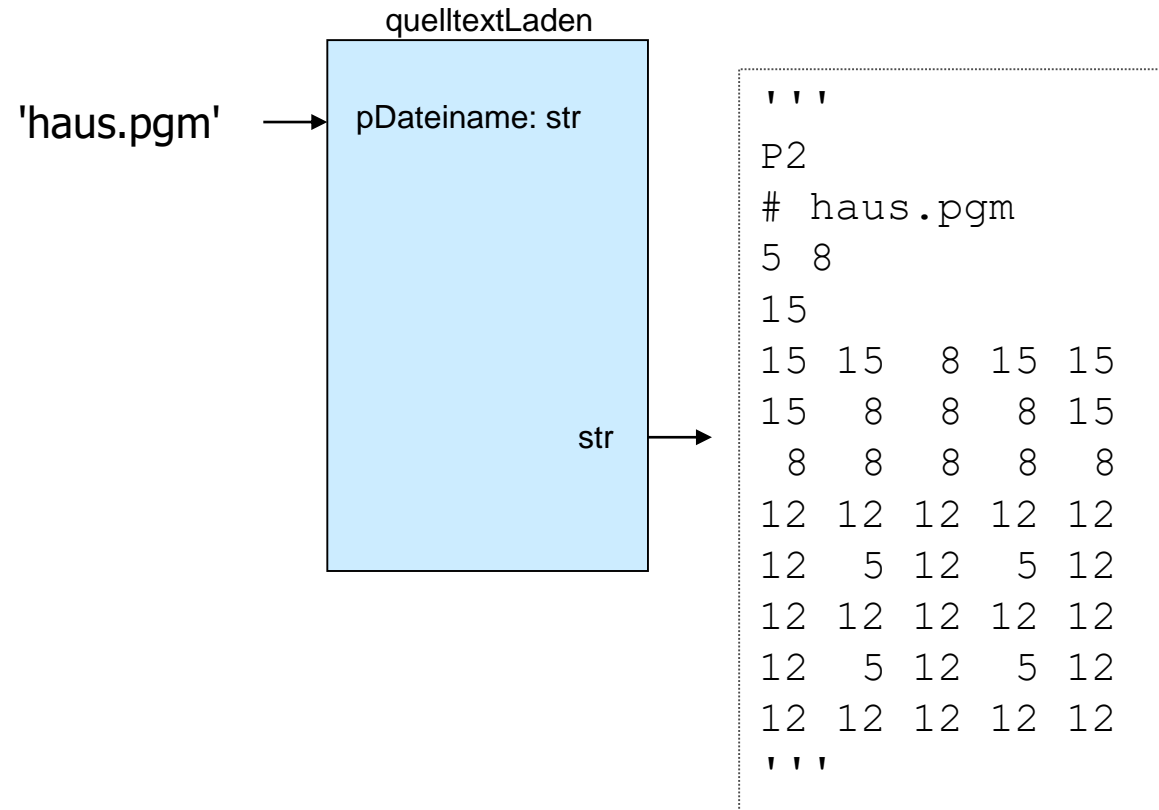
Teste die fertigen Funktionsbausteine  
(siehe [www.inf-schule.de/@/page/UPNmGEDRIQigzANh](http://www.inf-schule.de/@/page/UPNmGEDRIQigzANh)).

Alternativ: Entwickle selbst geeignete Funktionen, um die jeweiligen Positionen im Quelltext zu finden und erzeuge mit diesen Funktionen die strukturierten Daten zum Quelltext.

# Quelltext laden

## Problem 3:

Ein Quelltext soll aus einer Datei geladen werden.



```

def quelltextLaden(pDateiname):
    datei = open(pDateiname, 'r', encoding='iso-8859-1')
    quelltext = datei.read()
    datei.close()
    return quelltext
  
```

# Quelltext speichern

## Problem 4:

Ein Quelltext soll in einer Datei gespeichert werden.

'hausneu.pgm'

```
'''
P2
# haus.pgm neu
5 8
15
0 0 7 0 0 0 7 7 7 0
7 7 7 7 7 3 3 3 3 3
3 10 3 10 3 3 3 3 3
3 3 10 3 10 3 3 3 3
3 3
'''
```

quelltextSpeichern

pDateiname: str

pQuelltext: str

```
def quelltextSpeichern(pDateiname, pQuelltext):
    datei = open(pDateiname, 'w', encoding='iso-8859-1')
    datei.write(pQuelltext)
    datei.close()
```

## Ideen zur Grafikverarbeitung

Problem (\*):

Ein Graustufenbild im PGM-Format soll um einige Graustufen heller bzw. dunkler gemacht werden.

**Aufgabe:**

Entwickle geeignete Funktionen (Black-Box-Modellierung, Implementierung, Test der Implementierung) zur Lösung des Problems.

Problem (\*):

Aus einem Graustufenbild im PGM-Format soll ein Schwarz-weiß-Bild im PBM-Format erzeugt werden.

**Aufgabe:**

Entwickle geeignete Funktionen (Black-Box-Modellierung, Implementierung, Test der Implementierung) zur Lösung des Problems.

# PPM-Bild verarbeiten

Problem (\*\*):

Ein Farbbild im PPM-Format soll verarbeitet (z.B. invertiert) werden.

## **Aufgabe:**

Entwickle geeignete Funktionen (Black-Box-Modellierung, Implementierung, Test der Implementierung) zur Lösung des Problems.



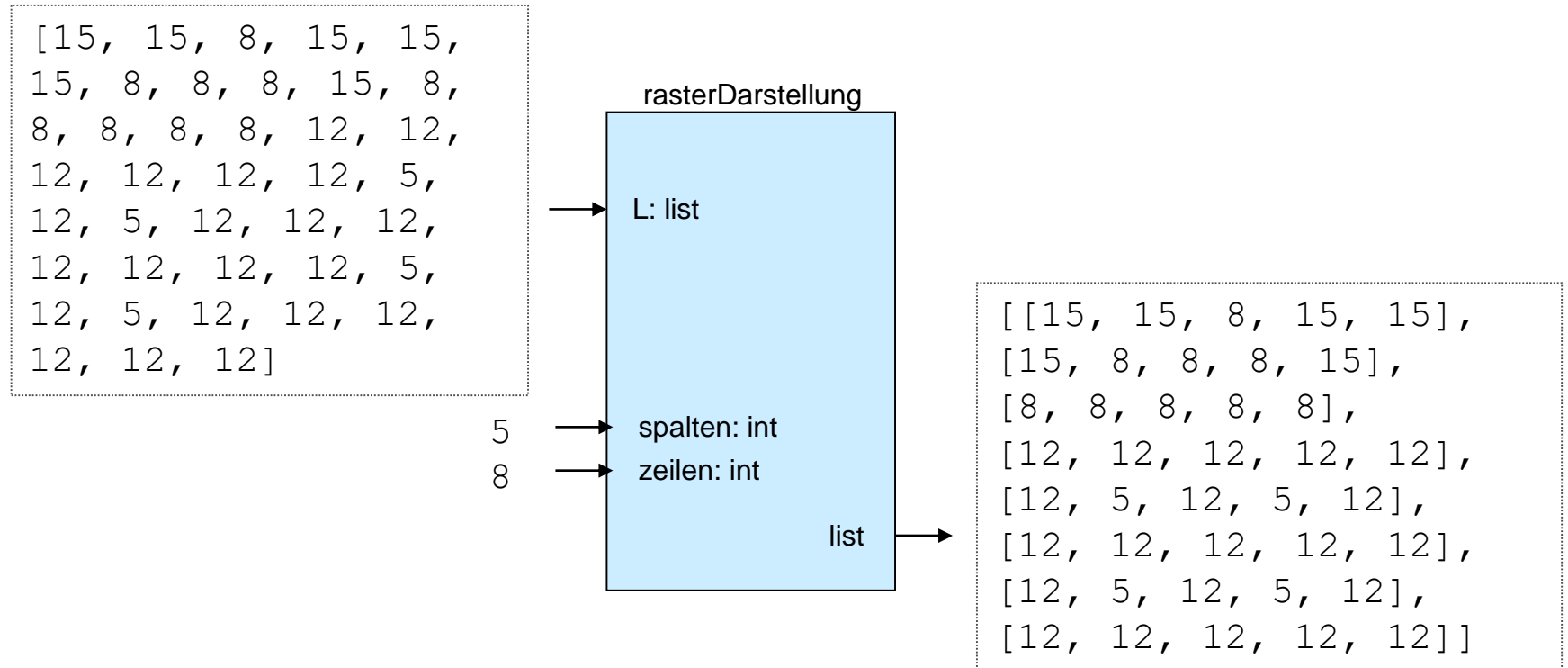
# Rasterdarstellung

Problem (\*):

Eine Listendarstellung der Pixel soll in eine Rasterdarstellung umgewandelt werden.

## Aufgabe:

Entwickle geeignete Funktionen (Black-Box-Modellierung, Implementierung, Test der Implementierung) zur Lösung des Problems.



Problem (\*\*\*):

Eine PGM-Grafik soll um  $90^\circ$  nach rechts / links gedreht werden.

**Aufgabe:**

Entwickle geeignete Funktionen (Black-Box-Modellierung, Implementierung, Test der Implementierung) zur Lösung des Problems.

**Aufgabe:**

Speichern Sie (oder mehrere) Funktion(en) zur Verarbeitung von Aktienkursen im Moodlekurs als Aufgabe ab.

