IQ.SH
Institut für Qualitätsentwicklung
an Schulen Schleswig-Holstein

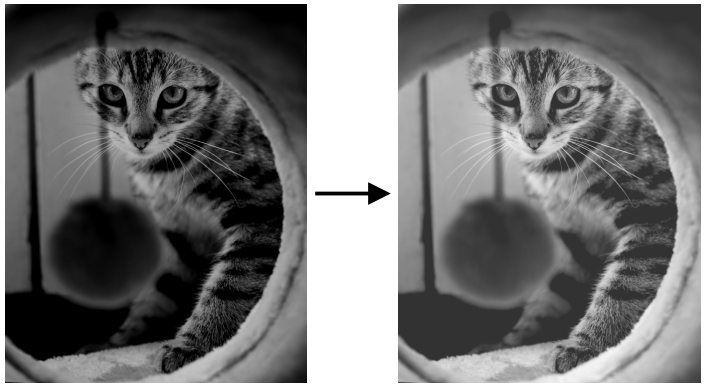
Bildverarbeitung mit Python

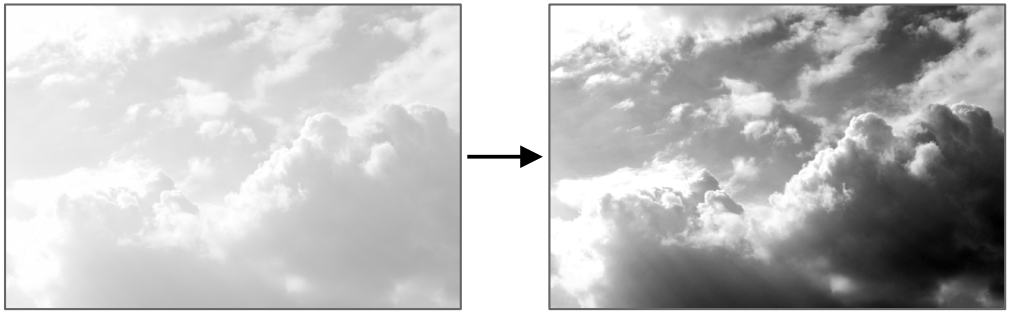
Textuelle Programmierung – Vertiefung

Warum Bildverarbeitung?

- Interessante Aufgabenstellungen, teils aus dem Alltag bekannt
- Bildverarbeitungsalgorithmen lassen sich auch „unplugged“ behandeln
- Motiviert durch visuelles Feedback und interessante Ergebnisse
- Gut geeignet für „algorithmisches Denken“:
  - Probleme lassen sich oft einfach lösen bzw. in einfachere zergliedern
  - Zweckmäßige und intuitive Einführung von Kontrollstrukturen und Unterprogrammen
  - Systematische Vorgehensweise beim Programmieren
  - Übung für objektbasierte Programmierung

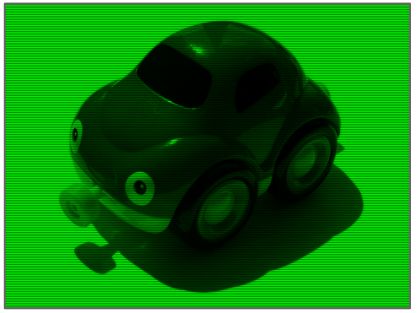
Beispiele





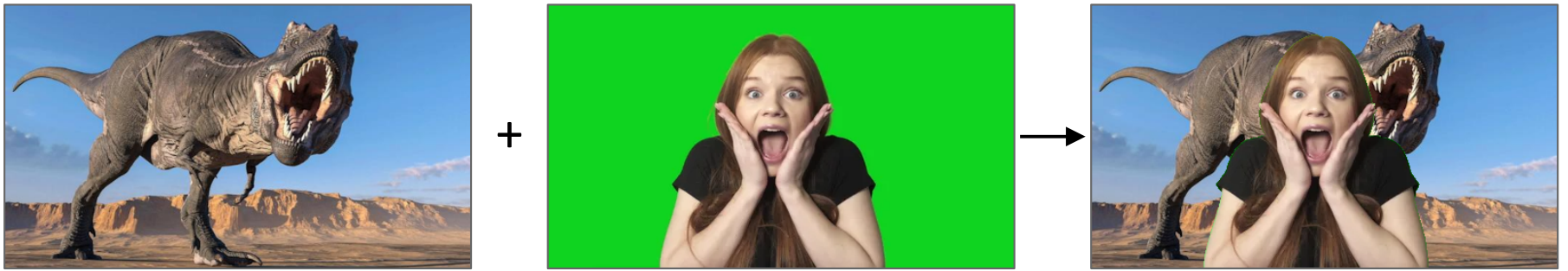
Helligkeits- und Kontrastanpassung

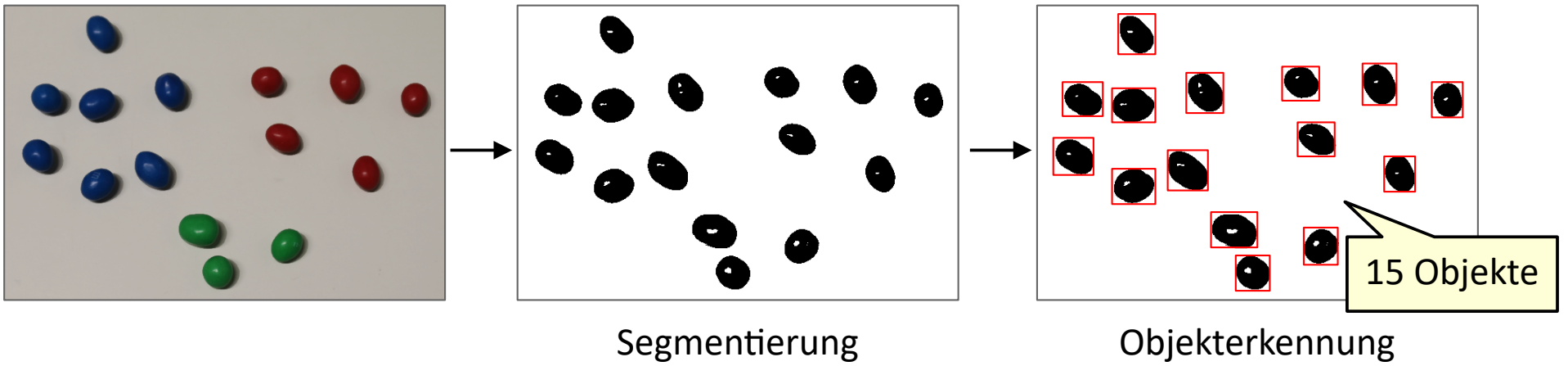
Linien zeichnen



Bildeffekte

Beispiele


+
Bildkomposition

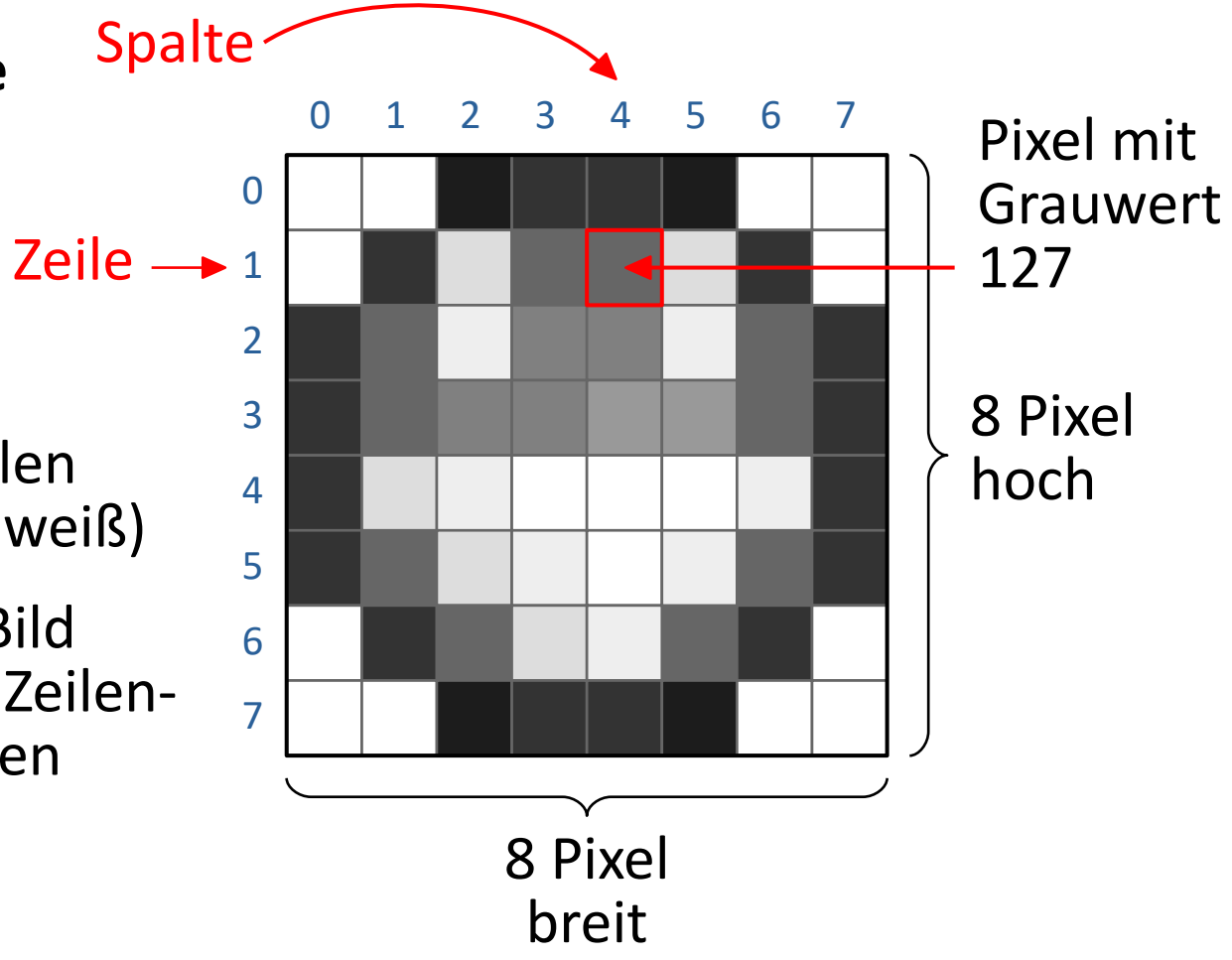

15 Objekte
Segmentierung
Objekterkennung

Inhalt

- Wiederholung: Rastergrafiken, Histogramme
- Bildverarbeitung mit Python und der Bibliothek „Pillow“
- Pixelweise Bildoperationen
  - Helligkeits-/Kontrastanpassung
  - Bildeffekte
  - Bildsegmentierung, Bildkomposition
- Grauwerthistogramme berechnen und interpretieren

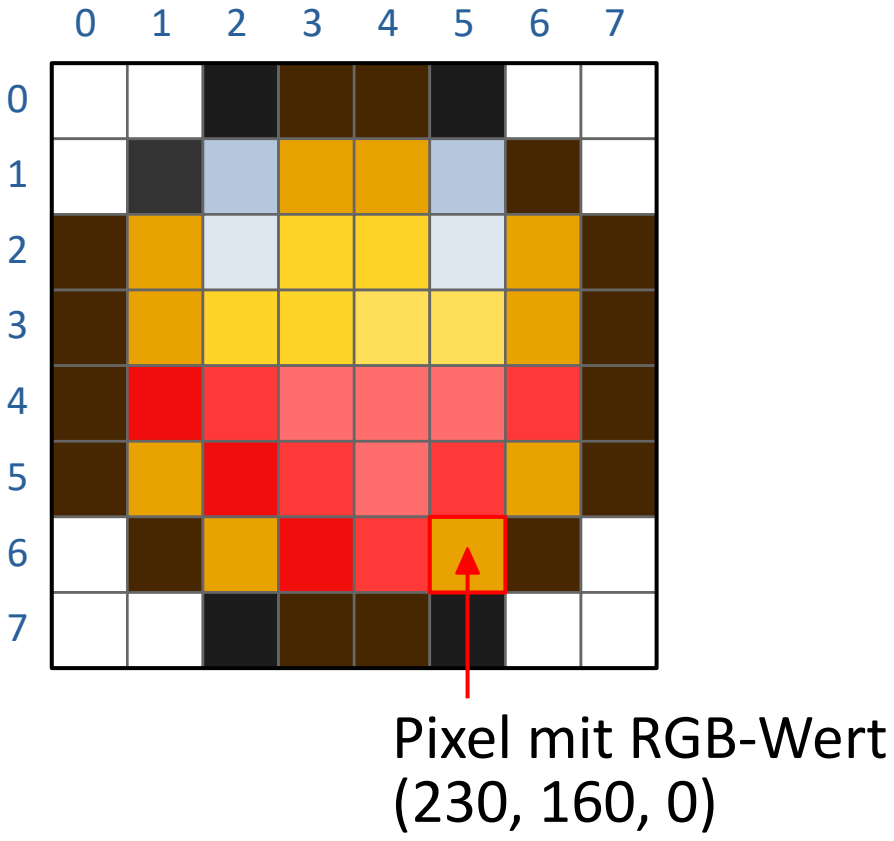
Wiederholung: Rastergrafiken

- Bild hat **Höhe** und **Breite** (in Pixeln angegeben)
- Jedes Pixel enthält einen diskreten Wert (z. B. 8-Bit-**Grauwert**)
- Grauwerte sind Ganzzahlen von 0 (schwarz) bis 255 (weiß)
- Position eines Pixels im Bild wird durch Spalten- und Zeilennummer (x, y) beschrieben (= Pixelkoordinaten)



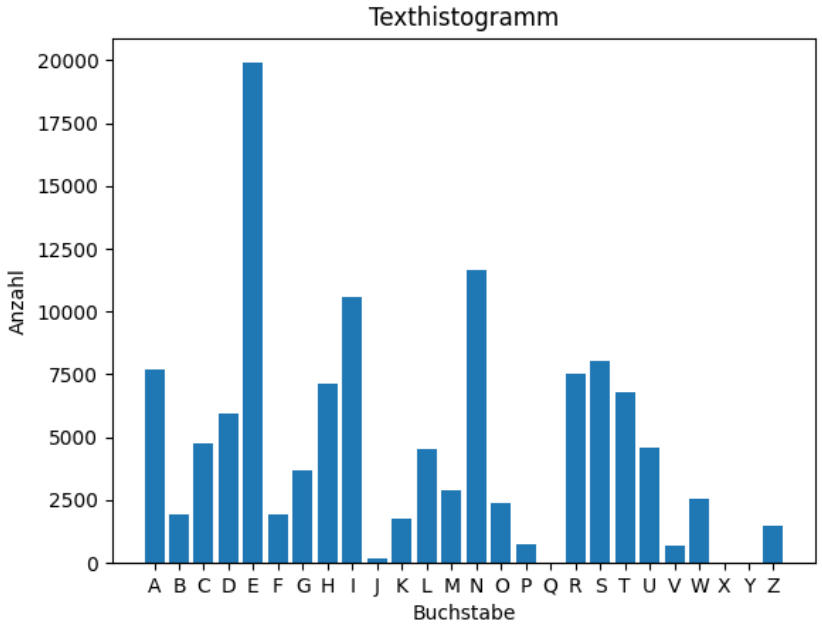
Wiederholung: Rastergrafiken

- Farbbilder im **RGB-Format** haben drei Werte pro Pixel
- RGB-Werte (8 Bit/Kanal) sind Ganzzahlen von 0 bis 255
- Alle Farben werden durch Rot-, Grün- und Blauwert „gemischt“ beschrieben
- Beispiel: (230, 160, 0) ist 90% rot, 63% grün, 0% blau (≈ dunkleres Gelb)

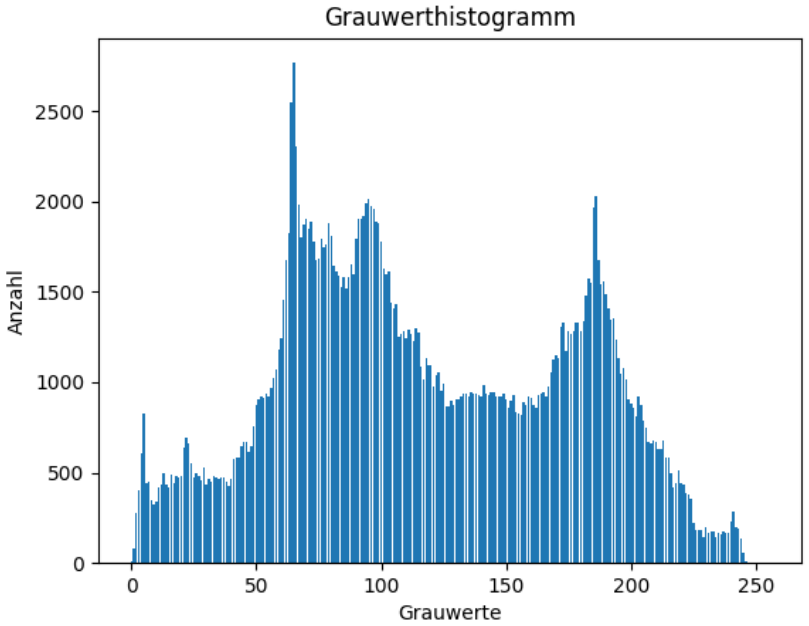


Wiederholung: Histogramme

- **Histogramme** sind grafische Darstellungen der Häufigkeitsverteilung von Werten in Datensätzen



Buchstabenhäufigkeit in Text



Grauwerthäufigkeit in Bild

Python-Bibliothek Pillow

- **Pillow** (PIL = Python Imaging Library) ist eine Python-Bibliothek, zum Öffnen, Bearbeiten und Speichern verschiedener Bilddateiformate
- Installation in Thonny über Menü „Extras“ → „Verwalte Pakete ...“

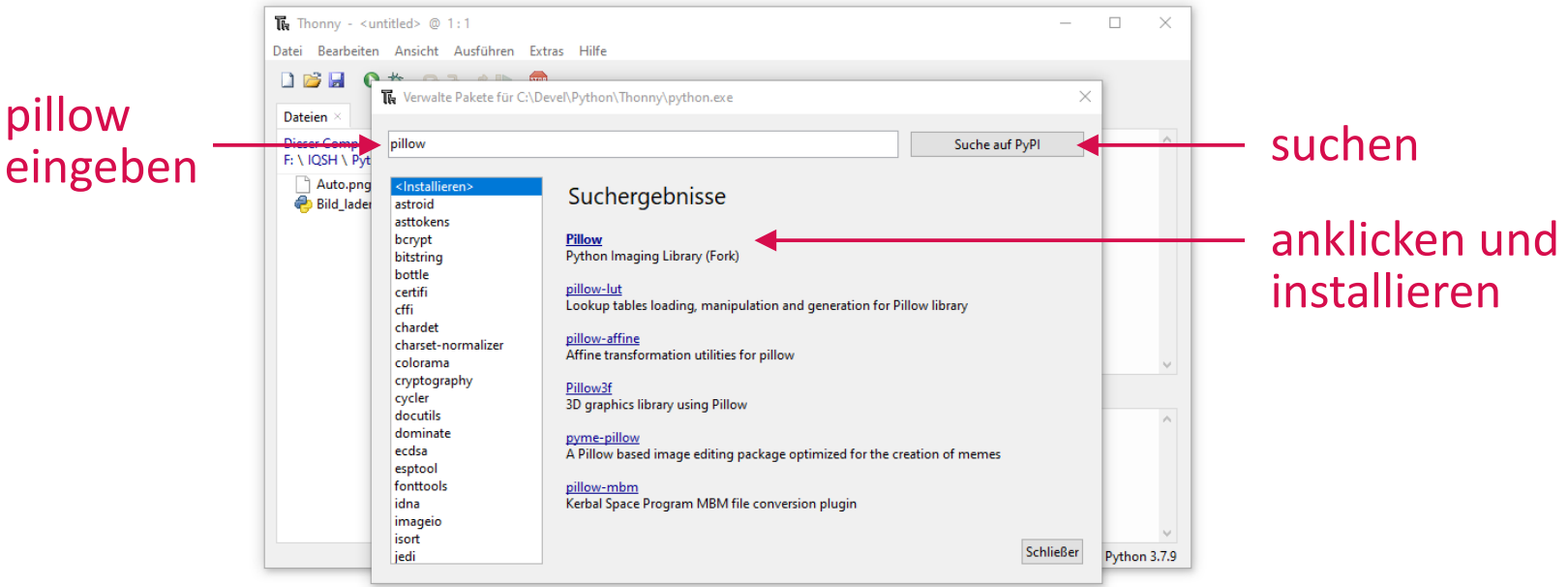


Bild laden und anzeigen

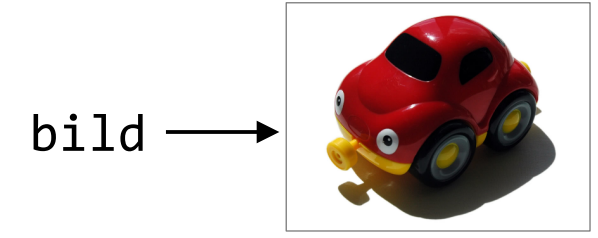
- Bibliothek importieren: `from PIL import Image`
- Bild laden: `bild = Image.open(`*Dateiname*`)`
  - Der Wert von *Dateiname* ist ein String, z. B. `'Auto.png'`
  - Bilddatei muss im selben Verzeichnis liegen wie das Python-Skript
- Bild anzeigen: `bild.show()`

```
from PIL import Image

bild = Image.open('Auto.png')
bild.show()
```

Bild laden und anzeigen

- Was passiert beim Ausführen von `bild = Image.open('Auto.png')`?
  1. Funktionsaufruf `Image.open('Auto.png')` liest Datei *Auto.png* und erstellt daraus ein Objekt der Klasse `Image` im Speicher
  2. Objektreferenz (= Verweis auf Objekt) wird in der Variablen `bild` gespeichert

```
from PIL import Image

bild = Image.open('Auto.png')
bild.show()
```



Attribute von Bild-Objekten

- `.width`, `.height` Breite und Höhe in Pixeln
- `.size` Bildgröße als Zweiertupel (Breite, Höhe)
- `.mode` Farbmodus (`'RGB'` für Farbe, `'L'` für Graustufen)

```
print('Breite:', bild.width, 'Pixel')
print('Höhe:', bild.height, 'Pixel')
print('insgesamt', bild.width * bild.height, 'Pixel')
if bild.mode == 'RGB':
    print('Farbmodus: RGB-Farbbild')
elif bild.mode == 'L':
    print('Farbmodus: Graustufenbild')
```

Farbbild in Graustufenbild umwandeln

- `.convert()` Wandelt Bilder zwischen verschiedenen Farbmodi um
- Beispiel: Erstelle ein Graustufenbild aus einem RGB-Farbbild:

```
graubild = farbbild.convert('L')
```

RGB-Farbbild (3 Kanäle)

Graustufenbild (1 Kanal)

Farbbild in Graustufenbild umwandeln

- Was passiert beim Ausführen von `graubild = farbbild.convert('L')`?
  1. Methodenaufruf `farbbild.convert('L')` erstellt neues Objekt der Klasse `Image` im Speicher ausgehend vom Objekt `farbbild`
  2. Neue Objektreferenz wird in der Variablen `graubild` gespeichert

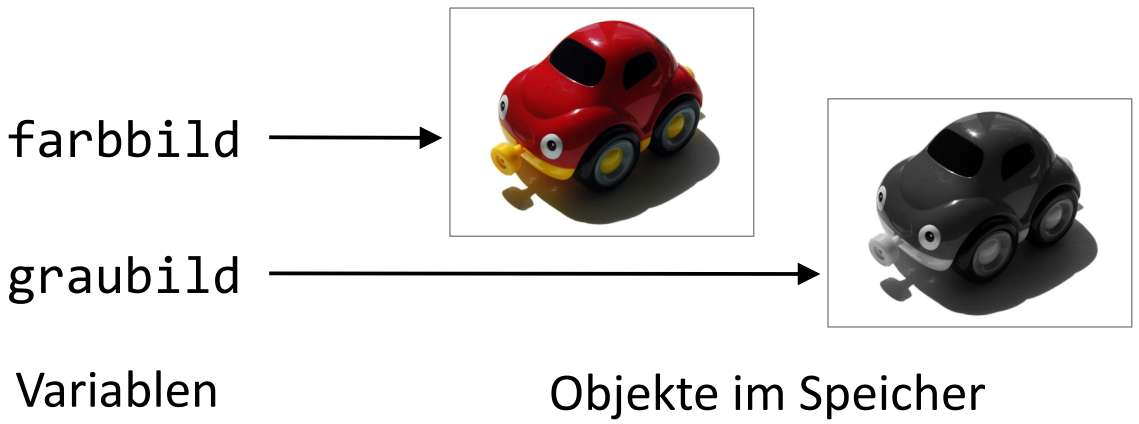


Bild skalieren

- `.resize((`*Breite*`,` *Höhe*`))` Ändert die Größe eines Bildes
- Beispiel: Skaliere ein Bild auf 240 × 240 Pixel Größe

```
kleines_bild = bild.resize((240, 240))
```

Originalbild (480 × 360)

skaliertes Bild (240 × 240 Pixel)

Pixelwerte lesen

- `.getpixel((x, y))` Fragt Pixelwert an Position (x, y) ab
- Ergebnis ist Ganzzahl 0, ..., 255 für Grauwertbild, z. B. 127
- Ergebnis ist Dreiertupel von Ganzzahlen 0, ..., 255 für RGB-Farbbild, z. B. `(128, 190, 255)`


```
wert = bild.getpixel((0, 0))
print('Wert des oberen linken Pixels:', wert)
wert = bild.getpixel((bild.width - 1, bild.height - 1))
print('Wert des unteren rechten Pixels:', wert)
```

Tupel in Python

- Der Datentyp `tuple` („Tupel“) ist in Python ähnlich wie eine Liste
- Tupel haben aber keine mutierenden Methoden (append, pop)
- x-y-Koordinaten und Bildgrößen werden als Zweiertupel (Paare) in runden Klammern angegeben, z. B. `(25, 100)`
- RGB-Farbwerte werden als Dreiertupel angegeben, z. B. `(0, 0, 100)` für dunkelblau oder `(255, 255, 0)` für gelb
- Tupel-Rückgabewerte lassen sich auch mehreren Variablen zuweisen:

```
r, g, b = bild.getpixel((0, 0))
```

Pixelwerte ändern

- `.putpixel((x, y), w)` Setze Pixel an Position (x, y) auf Wert w
- Farbwert ist Ganzzahl 0, ..., 255 für Grauwertbild
- Farbwert ist Dreiertupel von Ganzzahlen 0, ..., 255 für RGB-Farbbild

```
# Zeichne vertikale gelbe Linie in Bildmitte
for y in range(0, bild.height):
    bild.putpixel((bild.width // 2, y), (255, 255, 0))
# Zeichne horizontale dunkelblaue Linie in Bildmitte
for x in range(0, bild.width):
    bild.putpixel((x, bild.height // 2), (0, 0, 100))
```

Bilder pixelweise verarbeiten

- Typische Vorgehensweise bei Bildverarbeitung:
  - Gehe alle Pixel von oben nach unten, links nach rechts durch
  - Ändere oder lies jeden Pixelwert („mach etwas mit jedem Pixel“)

```
for y in range(0, bild.height):
    for x in range(0, bild.width):
        # Lies den Wert mit bild.getpixel((x, y))
        # und/oder ändere den Wert an dieser Stelle
        # mit bild.putpixel((x, y), ...)
```

Beispiel: Mittleren Grauwert berechnen

- Gehe alle Pixel von oben nach unten, links nach rechts durch
- Summiere alle Pixelwerte, teile am Ende durch Anzahl aller Pixel

```
g_summe = 0
for y in range(0, bild.height):
    for x in range(0, bild.width):
        g = bild.getpixel((x, y))
        g_summe = g_summe + g
g_mittel = g_summe / (bild.width * bild.height)
print('Mittlerer Grauwert:', g_mittel)
```

Beispiel: Größten Grauwert berechnen

- Gehe alle Pixel von oben nach unten, links nach rechts durch
- Vergleiche jeden Pixelwert mit bisher gefundenem größten Wert
  - Wenn Pixelwert größer ist, speichere ihn als neuen größten Wert

```
g_max = 0  # Beginne mit kleinstmöglichem Wert
for y in range(0, bild.height):
    for x in range(0, bild.width):
        g = bild.getpixel((x, y))
        if g > g_max:
            g_max = g
print('Maximaler Grauwert:', g_max)
```

Beispiel: Grauwertbild aufhellen/abdunkeln

- Gehe alle Pixel von oben nach unten, links nach rechts durch
- Addiere einen bestimmten Betrag („Offset“) zu jedem Pixelwert

Originalbild

alle Grauwerte um 50 erhöht

Beispiel: Grauwertbild aufhellen/abdunkeln

- Gehe alle Pixel von oben nach unten, links nach rechts durch
- Addiere einen bestimmten Betrag („Offset“) zu jedem Pixelwert

```
offset = 50
for y in range(0, bild.height):
    for x in range(0, bild.width):
        g = bild.getpixel((x, y))
        bild.putpixel((x, y), g + offset)
```

Pixelwerte werden beim Aufruf von `putpixel` automatisch auf min. 0 und max. 255 beschränkt

Beispiel: Farbbild aufhellen/abdunkeln

- Wie muss das Programm angepasst werden, wenn ein RGB-Farbbild statt eines Grauwertbildes bearbeitet wird?

```
offset = 50
for y in range(0, bild.height):
    for x in range(0, bild.width):
        r, g, b = bild.getpixel((x, y))
        bild.putpixel((x, y), (r + offset, g + offset, b + offset))
```

`getpixel` gibt Dreiertupel als Ergebnis zurück,
speichere die Werte hier in drei Variablen `r`, `g` und `b`

Beispiel: Automatische Helligkeitsanpassung

- Anpassung: Berechne Offset automatisch, so dass mittlerer Grauwert des Ausgabebildes ungefähr bei 127.5 liegt

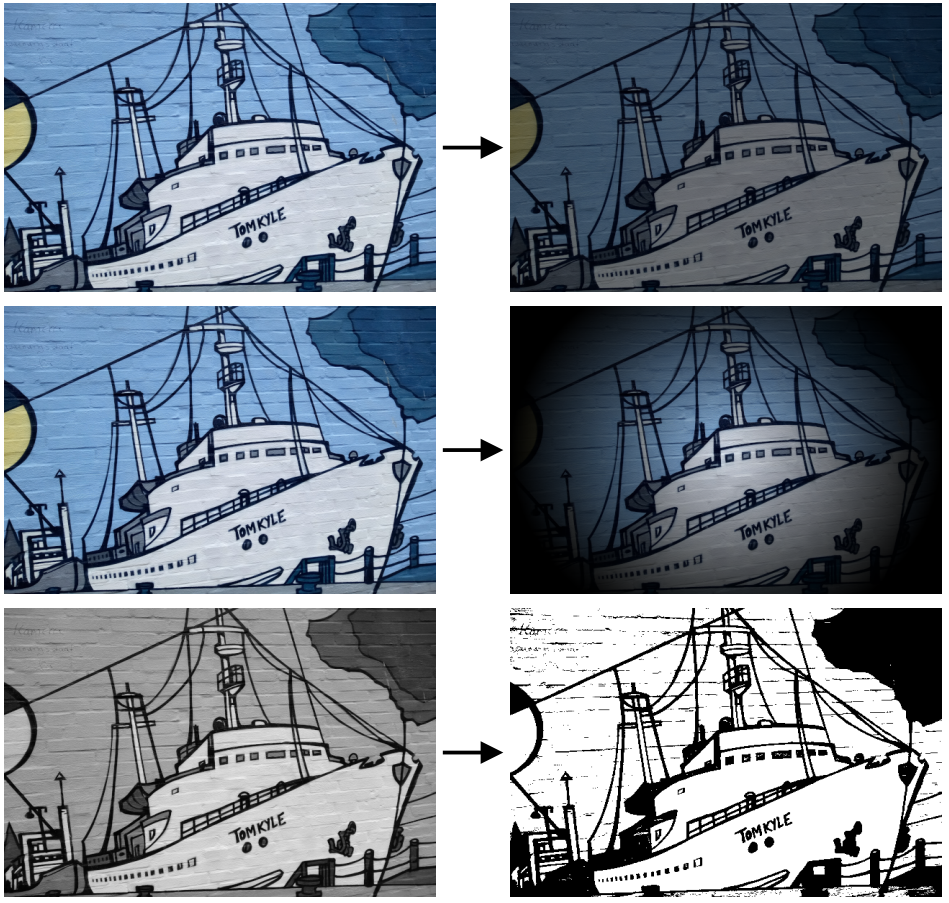
(Berechne `g_mittel` wie im Beispiel auf S. 23)

```
offset = round(127.5 - g_mittel)
for y in range(0, bild.height):
    for x in range(0, bild.width):
        g = bild.getpixel((x, y))
        bild.putpixel((x, y), g + offset)
```

Hinweis: Hier bietet es sich an, die Berechnung des mittleren Grauwertes in ein Unterprogramm auszulagern.

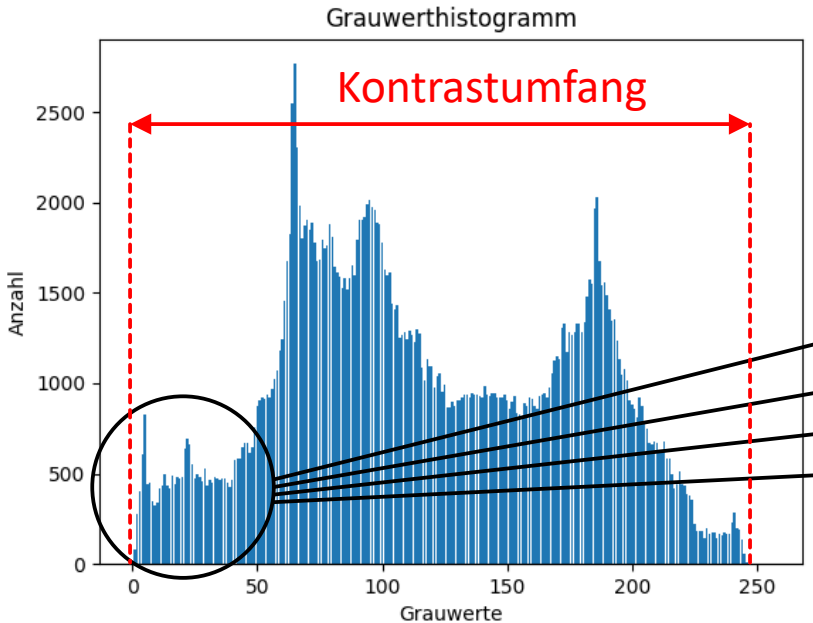
Weitere Beispiele zur pixelweisen Bildverarbeitung

- Bildkontrast halbieren:
  - neuer Wert = alter Wert / 2
- Vignettierung:
  - neuer Wert = alter Wert · (1 − Abstand zur Bildmitte / r)
- In Schwarzweißbild umwandeln:
  - Wenn alter Wert < 128: neuer Wert = 0, sonst: neuer Wert = 255



Grauwerthistogramme

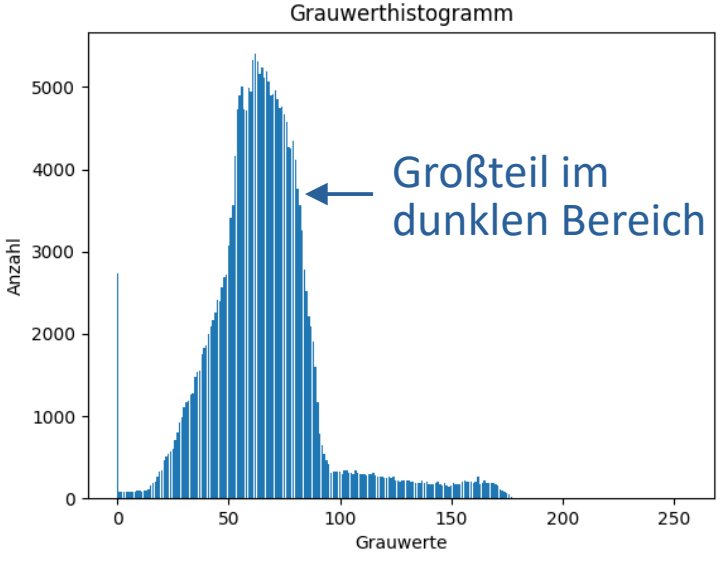
- Histogramm eines Grauwertbilds stellt für jeden Grauwert 0, ..., 255 dar, wieviele Pixel mit diesem Grauwert im Bild vorkommen

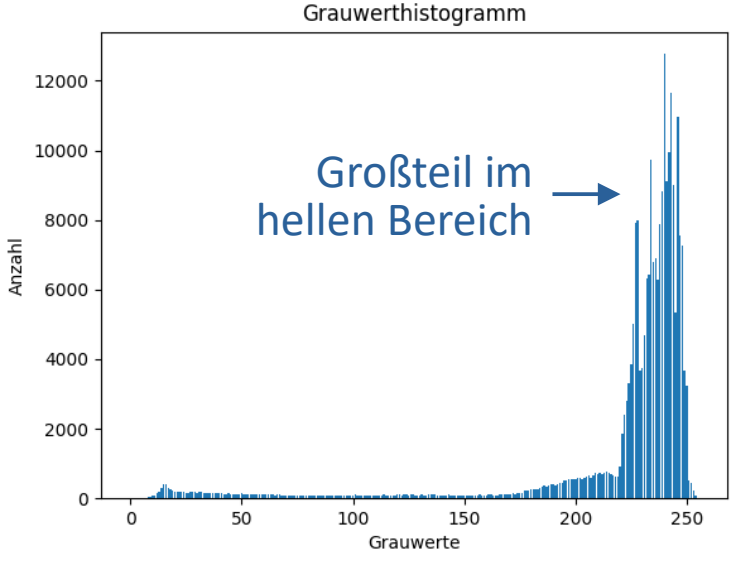


Grauwerthäufigkeit in Bild

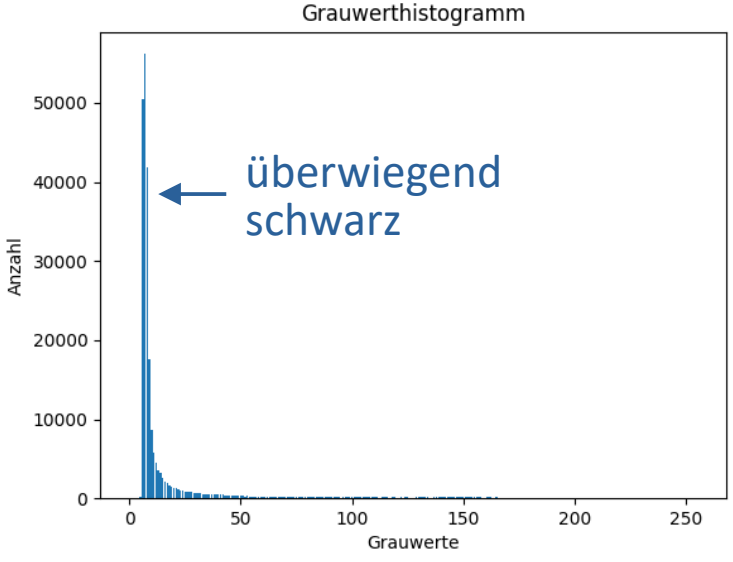
dunkle Bereiche

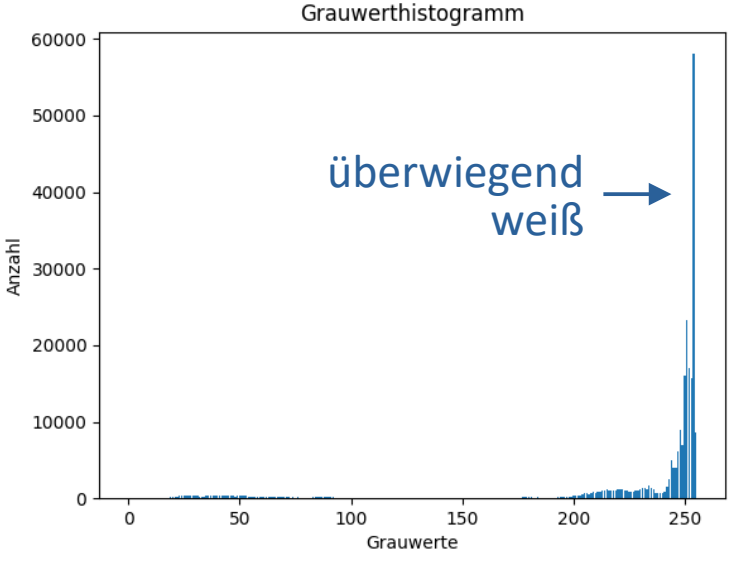
Beispiele: Grauwerthistogramme



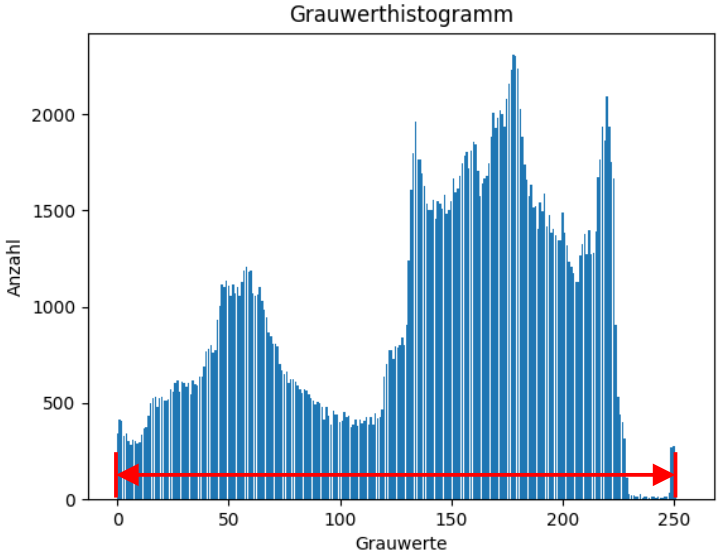


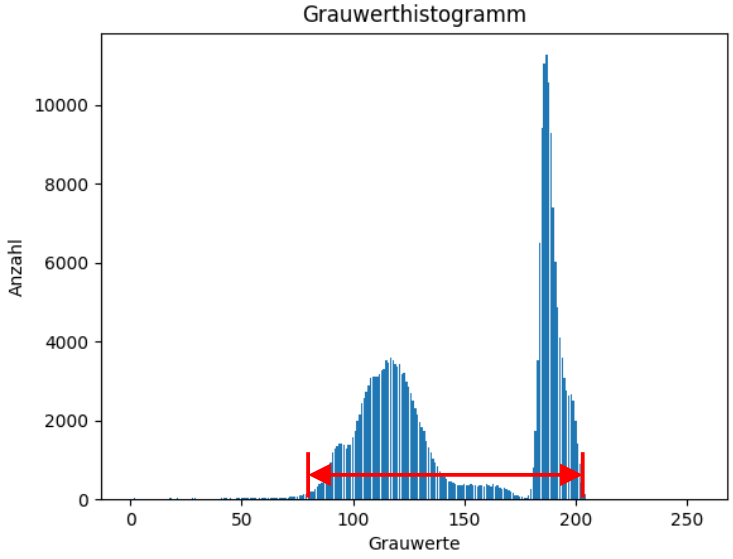
Beispiele: Grauwerthistogramme



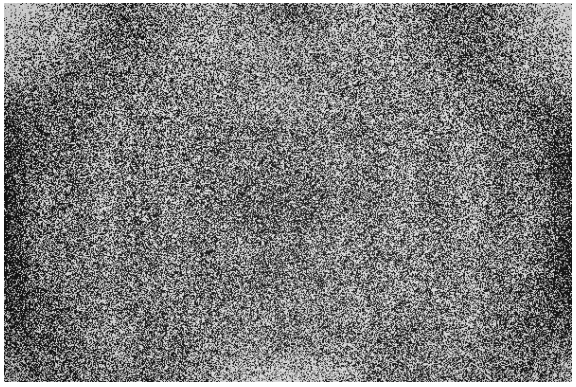


Beispiele: Grauwerthistogramme

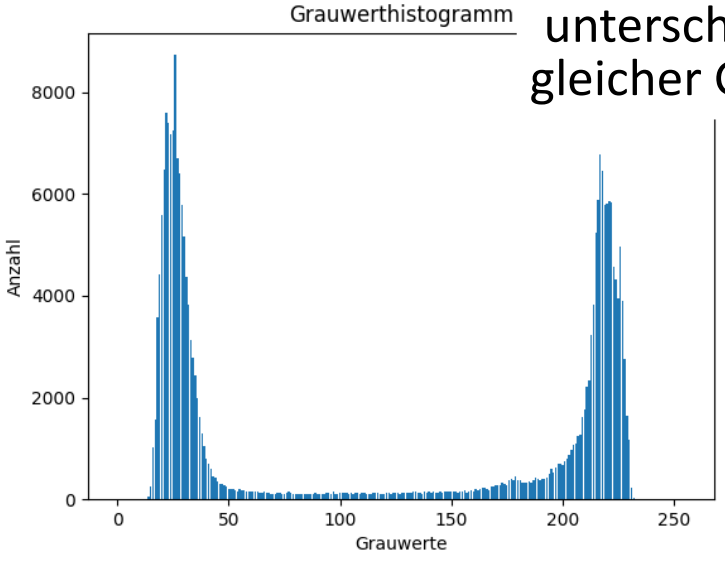


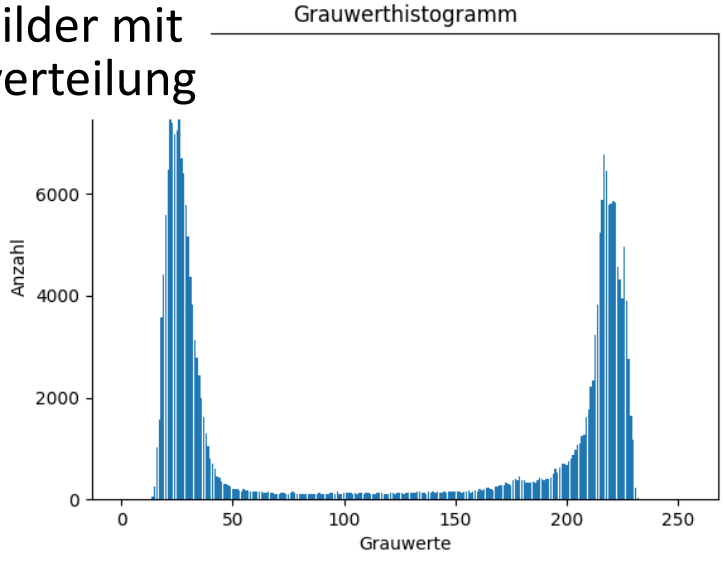


Beispiele: Grauwerthistogramme



unterschiedliche Bilder mit gleicher Grauwertverteilung





Grauwerthistogramm anzeigen

- 1. Möglichkeit: Bild mit Bildbearbeitungsprogramm öffnen (z. B. GIMP) und Histogramm anzeigen (in GIMP: Menü „Farben“ → „Werte...“)
- 2. Möglichkeit: Histogramm in Python berechnen und anzeigen
  - Berechne Liste **h** mit Histogrammwerten
  - Verwende Hilfsfunktion `Balkendiagramm.anzeigen`, um **h** als Balkendiagramm anzuzeigen (kopiere Datei *Balkendiagramm.py* aus Moodle vorher in das Arbeitsverzeichnis):

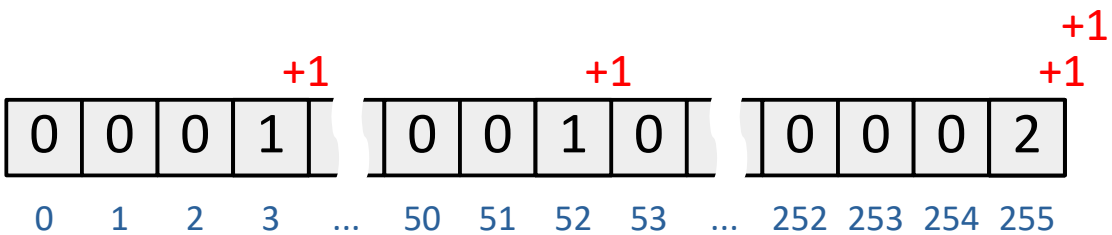
```
import Balkendiagramm
Balkendiagramm.anzeigen(h)
```

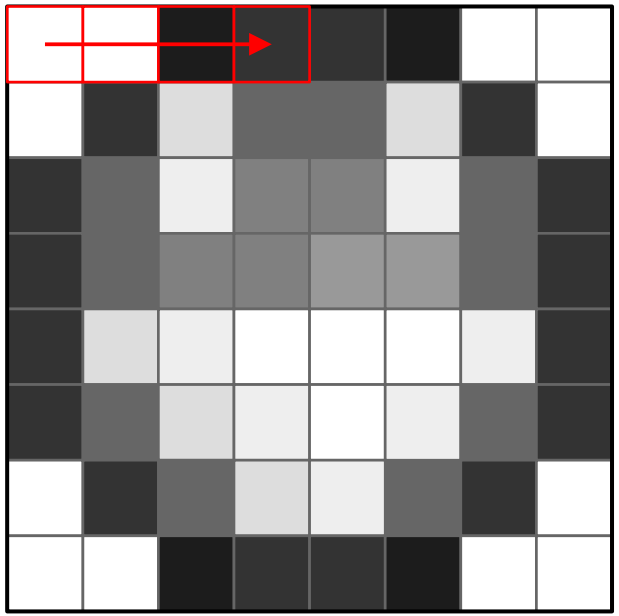
Wichtig: Installieren Sie zuvor in Thonny die Bibliothek matplotlib!

Grauwerthistogramm berechnen

- Erstelle Liste **h** mit 256 Nullen
- Gehe alle Pixelkoordinaten (x, y) durch:
  - Lies Grauwert g an Position (x, y)
  - Erhöhe **h** an Stelle g um 1:

    h[g] = h[g] + 1





Beispielprogramm: Grauwerthistogramm berechnen

```
from PIL import Image

bild = Image.open('Auto.png')
if bild.mode != 'L':
    bild = bild.convert('L')

h = [0] * 256
for y in range(0, bild.height):
    for x in range(0, bild.width):
        g = bild.getpixel((x, y))
        h[g] = h[g] + 1

import Balkendiagramm
Balkendiagramm.anzeigen(h)
```

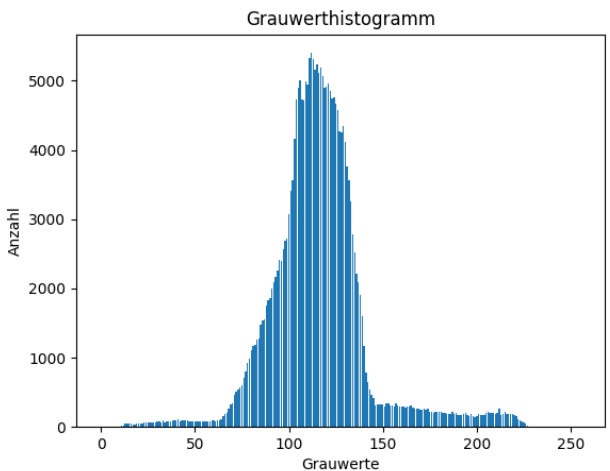
Bild laden und ggf. in Graustufen umwandeln

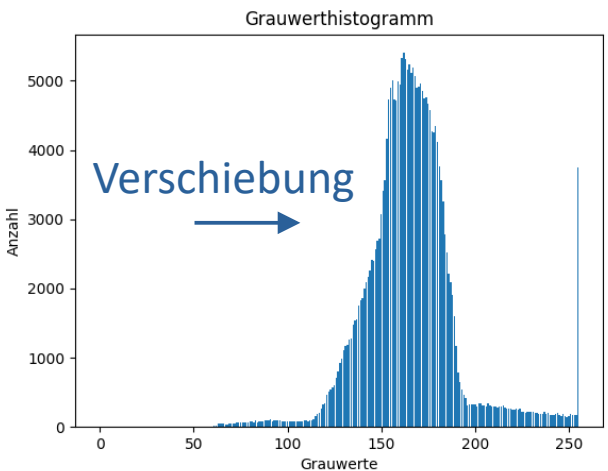
Histogramm berechnen

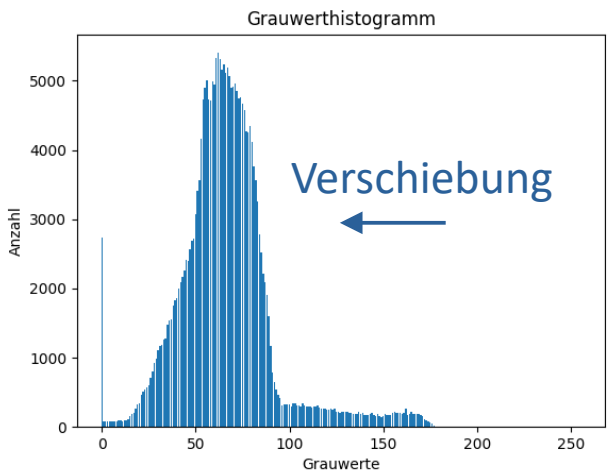
Alternativ:
`h = bild.histogram()`

Histogramm anzeigen

Änderung des Histogramms durch Bildoperationen

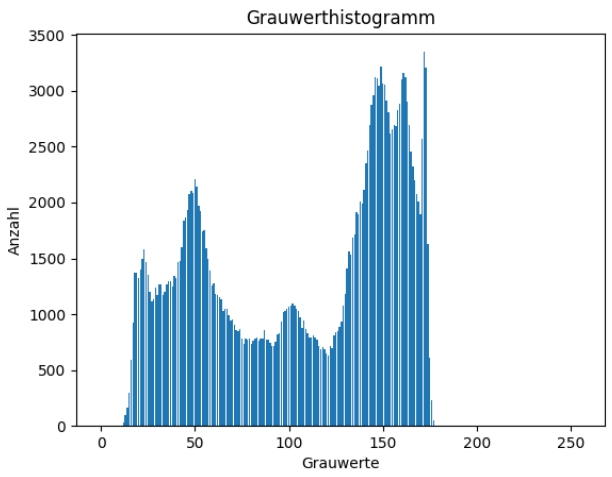


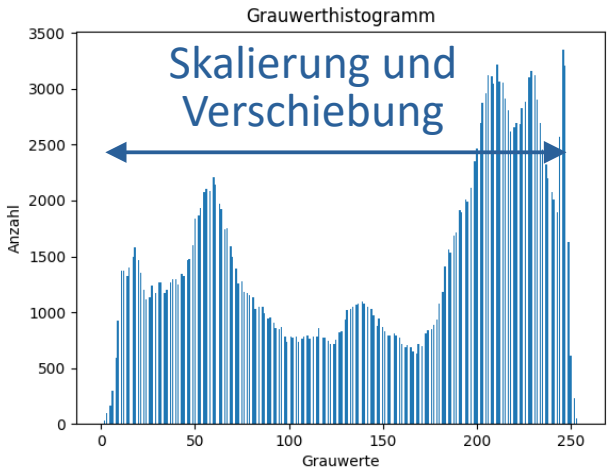




Änderung des Histogramms durch Bildoperationen

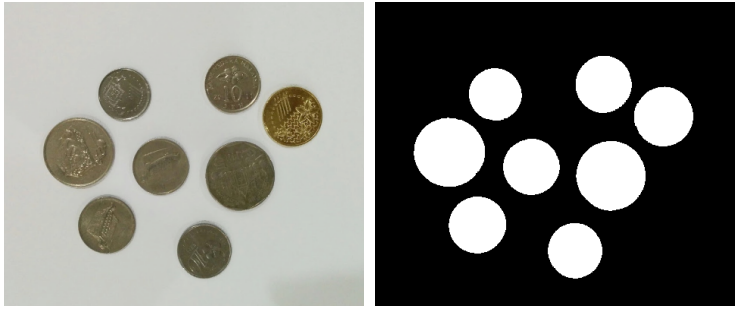
$$g_{neu} = 255 \cdot \frac{g_{alt} - g_{min}}{g_{max} - g_{min}}$$





Bildsegmentierung

- Wichtige Teilaufgabe in vielen Aufgaben der **Bilderkennung**: Unterscheide „Vordergrundpixel“ und „Hintergrundpixel“ im Bild
- Verschiedene Lösungsmöglichkeiten, z. B.:



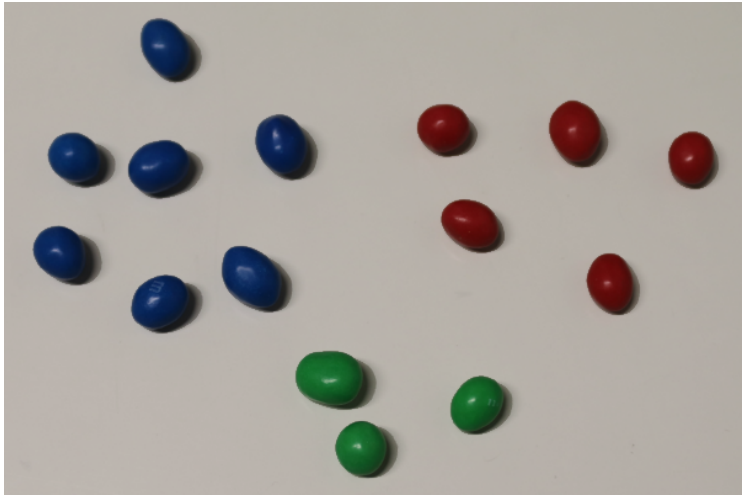
nach Helligkeit oder Farbe

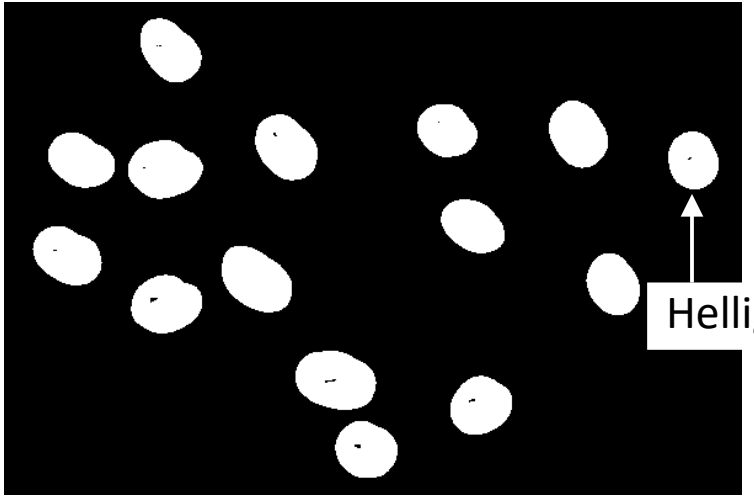
Vergleich mit einem Hintergrundbild

Bildsegmentierung nach Helligkeit

- Markiere dunklere Bereiche als „Vordergrund“ (hier weiß dargestellt) und hellere Bereiche als „Hintergrund“ (hier schwarz)
- Verwende Schwellenwert für Helligkeit zur Unterscheidung (z. B. 128)

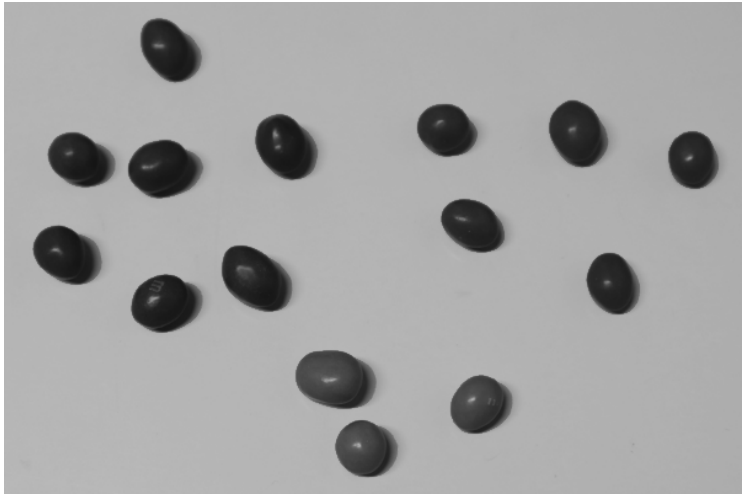
Formel: Helligkeit ≈ 0.3 · R + 0.6 · G + 0.1 · B

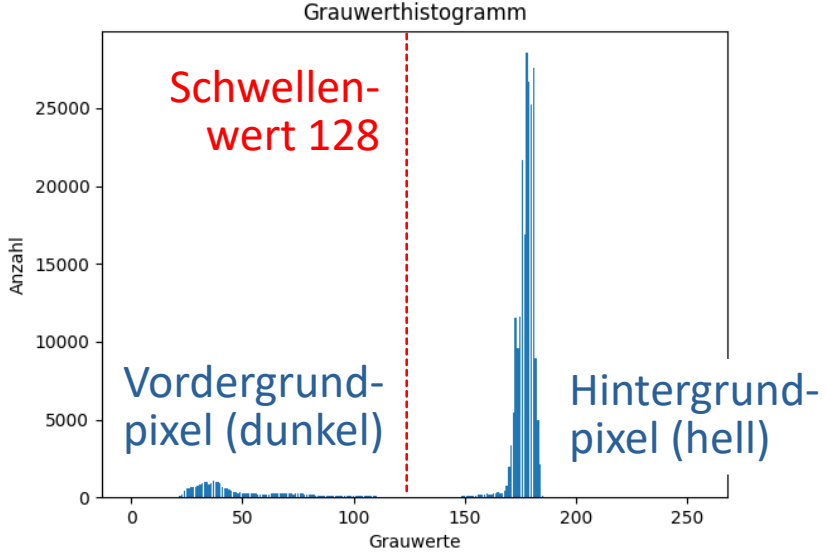




Bildsegmentierung nach Helligkeit

- Geeigneter Schwellenwert kann anhand des Grauwerthistogramms ermittelt werden → verwende Wert, der zwischen den beiden ausgeprägten hellen und dunklen Bereichen liegt



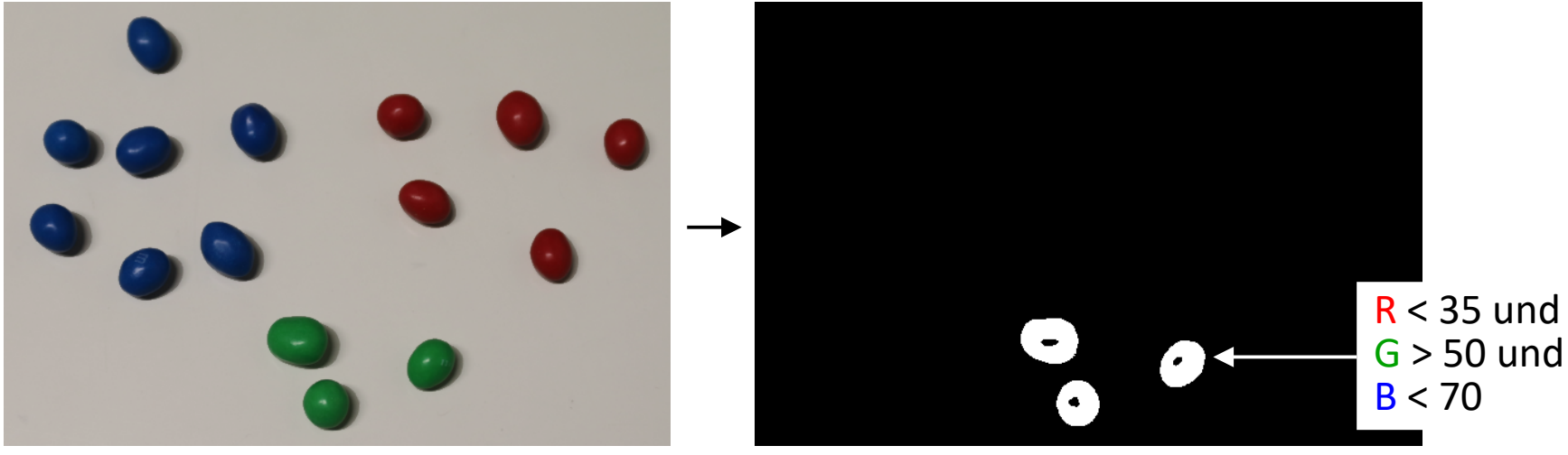


Bildsegmentierung nach Helligkeit

```
eingabe_bild = Image.open('Vordergrund.png')
ausgabe_bild = Image.new('L', eingabe_bild.size)

for y in range(0, eingabe_bild.height):
    for x in range(0, eingabe_bild.width):
        r, g, b = eingabe_bild.getpixel((x, y))
        grau = round(0.3*r + 0.6*g + 0.1*b)
        if grau < 128:
            ausgabe_bild.putpixel((x, y), 255)
        else:
            ausgabe_bild.putpixel((x, y), 0)

ausgabe_bild.show()
```

Bildsegmentierung nach Farbe

- Markiere Bereiche mit einer bestimmten Farbe als „Hintergrund“ oder als „Vordergrund“
- Verwende verschiedene Schwellenwerte für Rot-, Grün- und Blauwerte

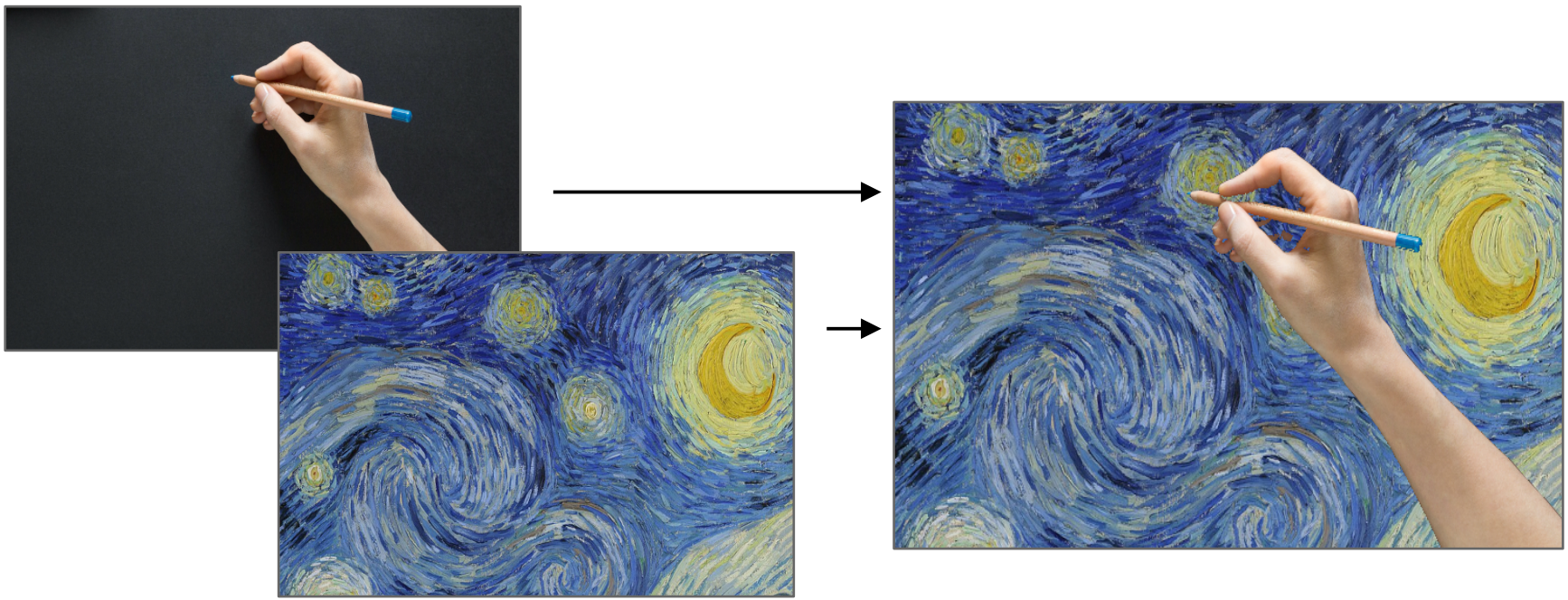


Bildsegmentierung nach Farbe

```
eingabe_bild = Image.open('Vordergrund.png')
ausgabe_bild = Image.new('L', eingabe_bild.size)

for y in range(0, eingabe_bild.height):
    for x in range(0, eingabe_bild.width):
        r, g, b = eingabe_bild.getpixel((x, y))
        if r < 35 and g > 50 and b < 70:
            ausgabe_bild.putpixel((x, y), 255)
        else:
            ausgabe_bild.putpixel((x, y), 0)

ausgabe_bild.show()
```

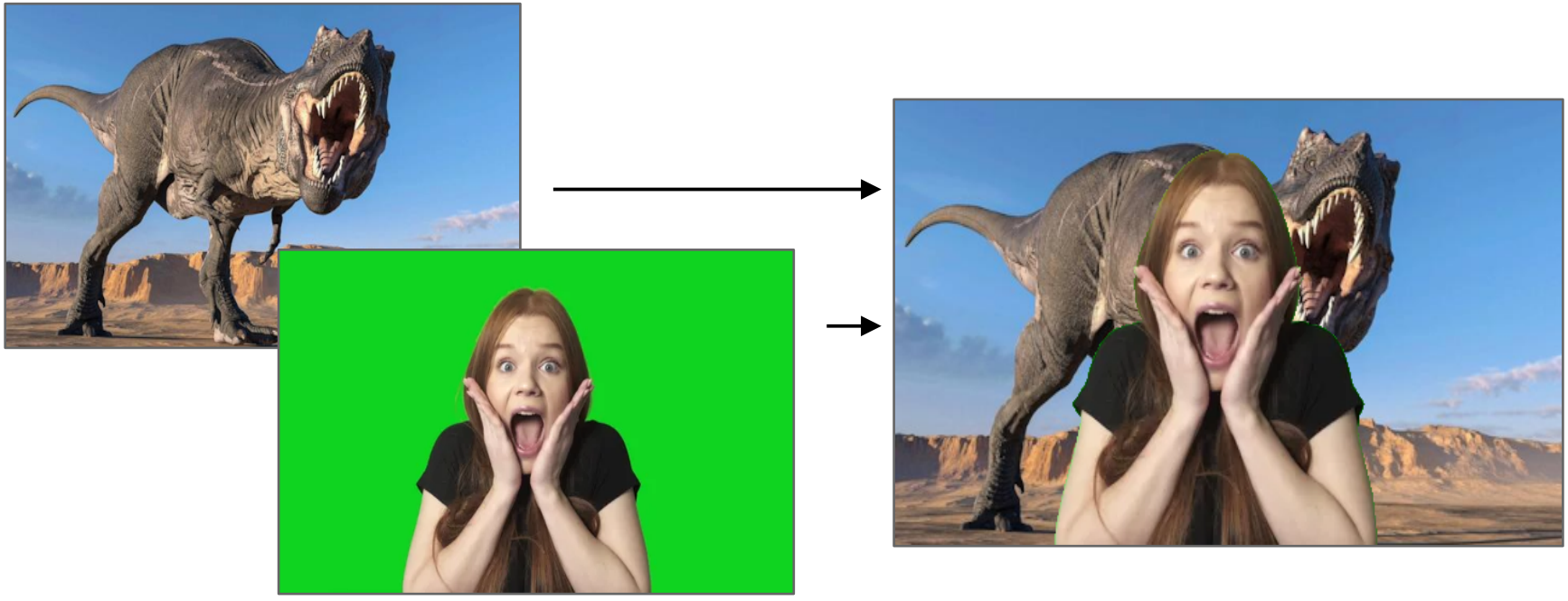
Hintergrund ersetzen

- Ersetze dunklere Bereiche („Hintergrund“) durch Pixelwerte aus einem anderen Bild (Hintergrundbild)



Chroma Keying

- Ersetze einen Farbbereich durch Pixelwerte aus einem Hintergrundbild (z. B. alle „grünen“ Pixel mit Rot < 128, Grün > 128, Blau < 128)



Chroma Keying

```
vordergrund = Image.open('Vordergrund.png')
hintergrund = Image.open('Hintergrund.png')
ausgabe_bild = Image.new('RGB', vordergrund.size)

for y in range(0, vordergrund.height):
    for x in range(0, vordergrund.width):
        r, g, b = vordergrund.getpixel((x, y))
        if r < 128 and g > 128 and b < 128:
            r, g, b = hintergrund.getpixel((x, y))
        ausgabe_bild.putpixel((x, y), (r, g, b))

ausgabe_bild.show()
```

Referenz: Grundlegende Methoden für Bilder

- Bild laden: `bild = Image.open(`*Dateiname*`)`
- Bild speichern: `bild.save(`*Dateiname*`)`
- Bild anzeigen: `bild.show()`
- Pixelwert abfragen: `farbwert = bild.getpixel((x, y))`
- Pixelwert ändern: `bild.putpixel((x, y), farbwert)`
- Was für eine Art von Wert ist `farbwert` bei `get-/putpixel`?
  - Für Grauwertbilder eine Ganzzahl
  - Für RGB-Farbbilder ein Tupel aus drei Ganzzahlen `(r, g, b)`

Referenz: Grundlegende Methoden für Bilder

- Graustufenbild aus RGB-Farbbild erzeugen:

```
graubild = farbbild.convert('L')
```

- RGB-Farbbild aus Graustufenbild erzeugen:

```
farbbild = graubild.convert('RGB')
```

- Neues, leeres RGB-Farbbild oder Graustufenbild erzeugen (`farbmodus` ist `'RGB'` oder `'L'`):

```
neues_bild = Image.new(farbmodus, (breite, hoehe))
```

- Hinweis: Diese Methoden liefern ein *neues* Bild-Objekt zurück.

Referenz: Weitere Methoden für Bilder

- Bild skalieren: `neues_bild = bild.resize((`*Breite*`, `*Höhe*`))`
- Bild ausschneiden: `ausschnitt = bild.crop((`*x1*`, `*y1*`, `*x2*`, `*y2*`))`
- Bild drehen (Winkel in Grad ↺): `neues_bild = bild.rotate(`*Winkel*`)`
- Hinweis: Diese Methoden liefern ein *neues* Bild-Objekt zurück.

- Bild einfügen: `bild.paste(`*anderes Bild*`, (`*x1*`, `*y1*`, `*x2*`, `*y2*`))`
- Hinweis: Diese Methode *verändert* das Bild-Objekt `bild`.

Referenz: RGB-Farbwerte

Rot:	(255, 0, 0)
Grün:	(0, 255, 0)
Blau:	(0, 0, 255)
Gelb:	(255, 255, 0)
Magenta:	(255, 0, 255)
Cyan:	(0, 255, 255)
Weiß:	(255, 255, 255)
Grau:	(127, 127, 127)
Schwarz:	(0, 0, 0)

Links und Quellenangaben

- Offizielle Dokumentation zu Pillow (engl.)
  - https://pillow.readthedocs.io
- Kapitel „Digitale Bildverarbeitung“ im Online-Skript, Sek. I+II
  - https://weiterbildung-informatik-intern.wollw.de/sek2-chapter19

- Quellen der verwendeten Bilder (lizenzfrei):
  - https://www.pexels.com
  - https://unsplash.com
  - https://www.publicdomainpictures.net