Python-Kurs.eu – Bildverarbeitung mit Python

Von Pixeln zu praktischen Verfahren

Bernd Klein

14. August 2026

Python-Kurs.eu – Bildverarbeitung mit Python

Von Pixeln zu praktischen Verfahren

Autor: Bernd Klein
Website: https://www.python-kurs.eu
Stand dieser Ausgabe: 14. August 2026

Diese Ausgabe wurde automatisiert aus den autoritativen Jupyter-Notebooks von `python-kurs.eu` erzeugt. Satz und technische Produktion erfolgen reproduzierbar mit Python, Pandoc, LuaLaTeX und - soweit erforderlich - PythonTeX.

Die Online-Fassung kann aktueller sein als diese PDF-Ausgabe. Trotz sorgfältiger Prüfung kann für die Fehlerfreiheit von Texten und Programmen keine Gewähr übernommen werden.

Die im Anhang beworbenen Verlagstitel sind eigenständige Bücher beim Carl Hanser Verlag. Die vorliegende automatisch erzeugte Ausgabe ist **keine Veröffentlichung des Carl Hanser Verlags**.

Das Python-Logo wird zur Kennzeichnung des Bezugs zur Programmiersprache Python verwendet. Python und das Python-Logo sind Marken der Python Software Foundation.

Inhaltsverzeichnis

1 Python Bildverarbeitung **1**
1.0.1 Bildverarbeitung 1

2 Bildverarbeitungs Techniken **19**
2.0.1 Techniken der Bildverarbeitung 19

3 Video Aus Bildern Erzeugen **32**
3.0.1 Videos aus Bildern erstellen 32

4 Videos Bewegliche Objekte **40**
4.0.1 Videos mit beweglichen Objekten 40
4.0.2 Zufällige Kachel (tile) im Bild 44
4.0.3 Wasserzeichen mit zusätzlichem Bild 46
4.0.4 Wasserzeichen auf Kachel 47
4.0.5 Kachel nahe beieinandeer 50
4.0.6 Bilder mit beweglichen Wasserzeichen 51

Weitere Bücher von Bernd Klein **55**

1 Python Bildverarbeitung

```
# invisible
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 300
```

1.0.1 Bildverarbeitung

Einführung

Es war noch nie einfacher, ein Bild aufzunehmen als heute. Alles was Sie normalerweise brauchen ist ein Handy. Dies sind die Grundvoraussetzungen, um ein Bild aufzunehmen und anzusehen. Das Fotografieren ist kostenlos, wenn wir die Kosten für das Mobiltelefon nicht berücksichtigen, das ohnehin oft für andere Zwecke gekauft wird. Vor einer Generation brauchten Amateur- und echte Künstler spezielle und oft teure Ausrüstung, und die Kosten pro Bild waren alles andere als kostenlos.

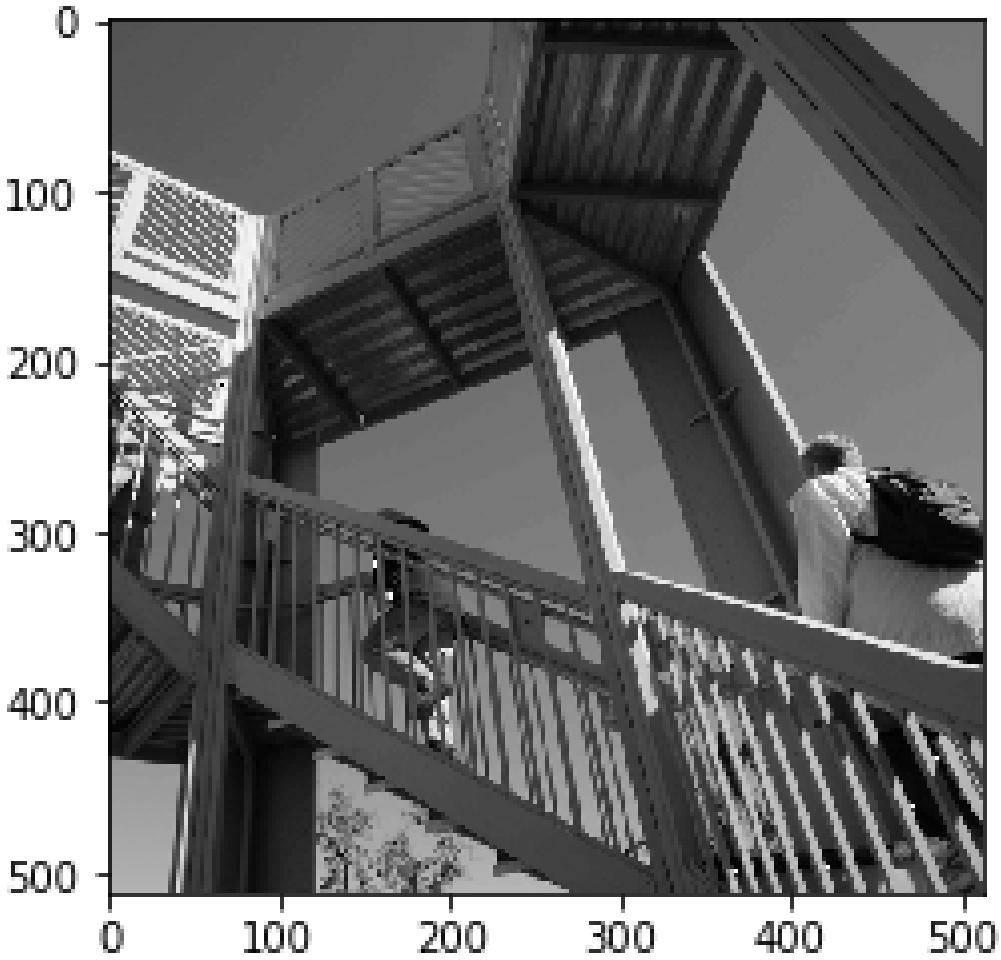
Wir machen Fotos, um großartige Momente unseres Lebens in der Zeit zu konservieren. "Eingelegte Erinnerungen", die bereit sind, in Zukunft nach Belieben "geöffnet" zu werden.

Ähnlich wie beim Einmachen müssen wir die richtigen Konservierungsmittel verwenden. Natürlich bietet uns das Mobiltelefon auch eine Reihe von Bildverarbeitungssoftware, aber sobald wir eine große Anzahl von Fotos verarbeiten müssen, benötigen wir andere Tools. Hier kommen Programmierung und Python ins Spiel. Python und seine Module wie Numpy, Scipy, Matplotlib und andere Spezialmodule bieten die optimale Funktionalität, um mit der Flut von Bildern fertig zu werden.

Um Ihnen das notwendige Wissen zu vermitteln, befasst sich dieses Kapitel unseres Python-Tutorials mit der grundlegenden Bildverarbeitung und -manipulation. Zu diesem Zweck verwenden wir die Module NumPy, Matplotlib und SciPy.

Wir beginnen mit dem `scipy` Paket `misc`. Die Hilfedatei besagt, dass scipy.misc verschiedene Dinge enthalte, die man sonstwo nicht unterbringen konnte. So enthält es auch beispielsweise ein paar Bilder, wie das folgende:

```
from scipy import misc
import matplotlib.pyplot as plt

ascent = misc.ascent()
plt.gray()
plt.imshow(ascent)
plt.show()
```



Zusätzlich zum Bild erkennen wir, dass die Ticks an den Achsen ebenfalls ausgegeben werden. Das ist sicherlich interessant, wenn Sie Informationen zur Größe und Pixel-Position benötigen, jedoch möchte man in den meisten Fällen das Bild ohne diese Informationen sehen. Indem wir den Befehl plt.axis(“off”) hinzufügen, können wir die Ticks und Achsen ausblenden:

```
from scipy import misc

ascent = misc.ascent()
import matplotlib.pyplot as plt
plt.axis("off") # removes the axis and the ticks
plt.gray()
plt.imshow(ascent)
plt.show()
```

Wir sehen, dass der Typ des Bildes ein Integer-Array ist:

```
ascent.dtype
```

```
dtype('int64')
```

Wir können auch die Größe des Bildes prüfen:

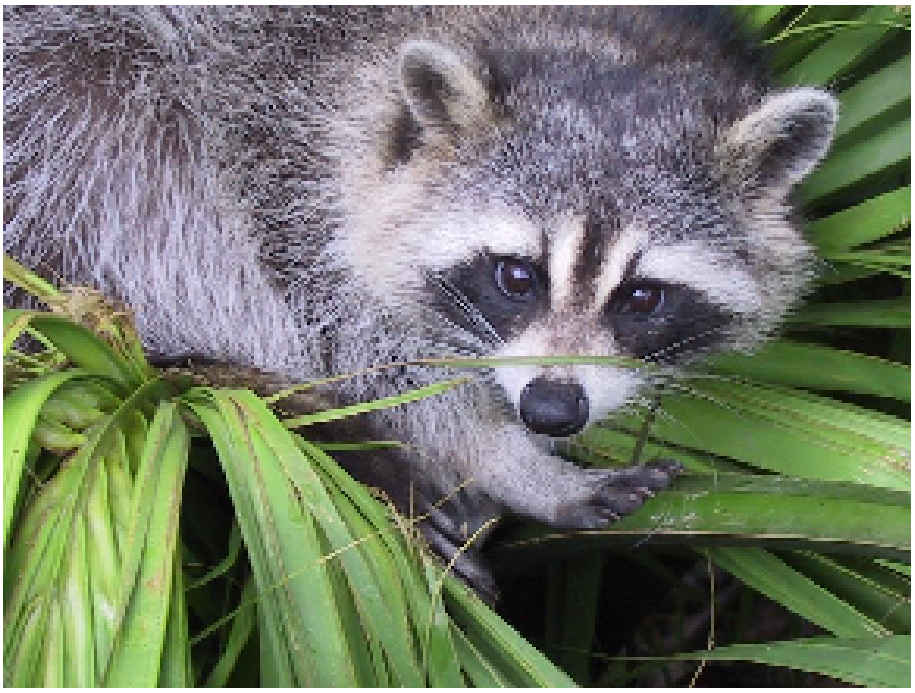
```
ascent.shape
```

```
(512, 512)
```

Das misc-Packet beinhaltet auch ein Bild eines Waschbären:

```
import scipy.misc
face = scipy.misc.face()
print(face.shape)
print(face.max)
print(face.dtype)
plt.axis("off")
plt.gray()
plt.imshow(face)
plt.show()
```

```
(768, 1024, 3)
<built-in method max of numpy.ndarray object at 0x7fd05e484f30>
uint8
```



```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

Matplotlib unterstützt nur PNG-Bilder.

```
img = mpimg.imread('frankfurt.png')
```

```
print(img[:3])
```

```
[[[0.4117647  0.5686275  0.8       ]
  [0.40392157 0.56078434 0.7921569 ]
  [0.40392157 0.5686275  0.79607844]
  ...
  [0.48235294 0.62352943 0.81960785]
  [0.47843137 0.627451   0.81960785]
  [0.47843137 0.62352943 0.827451  ]]

 [[0.40784314 0.5647059  0.79607844]
  [0.40392157 0.56078434 0.7921569 ]
  [0.40392157 0.5686275  0.79607844]
  ...
  [0.48235294 0.62352943 0.81960785]
  [0.47843137 0.627451   0.81960785]
  [0.48235294 0.627451   0.83137256]]
```

```
[[0.40392157 0.5686275  0.79607844]
 [0.40392157 0.5686275  0.79607844]
 [0.40392157 0.5686275  0.79607844]
 ...
 [0.48235294 0.62352943 0.81960785]
 [0.48235294 0.62352943 0.81960785]
 [0.4862745  0.627451   0.83137256]]]
```

```
plt.axis("off")
imgplot = plt.imshow(img)
```

```
lum_img = img[:,:,1]
print(lum_img)
```

```
[[0.5686275  0.56078434 0.5686275  ... 0.62352943 0.627451   0.62352943]
 [0.5647059  0.56078434 0.5686275  ... 0.62352943 0.627451   0.627451  ]
 [0.5686275  0.5686275  0.5686275  ... 0.62352943 0.62352943 0.627451  ]
 ...
 [0.31764707 0.32941177 0.32941177 ... 0.30588236 0.3137255  0.31764707]
 [0.31764707 0.3137255  0.32941177 ... 0.3019608  0.32156864 0.3372549 ]
 [0.31764707 0.3019608  0.33333334 ... 0.30588236 0.32156864 0.33333334]]
```

```
plt.axis("off")
imgplot = plt.imshow(lum_img)
```

Färbung, Schatten und Farbton

Jetzt möchten wir auf die Tönung eines Bildes eingehen. Tönung ist ein Ausdruck aus der Farb-Theorie ist eine oft von Malern verwendete Technik. Es ist kaum vorstellbar, wenn man an Maler denkt, gleichzeitig nicht an die Niederlande zu denken. Wir verwenden ein Bild mit "Holländischen Windmühlen" in unserem nächsten Beispiel. (Das Bild wurde in Kinderdijk aufgenommen, ein Dorf in den Niederlanden über 15km östlich von Rotterdam und über 50km entfernt von Den Haag. Es ist ein UNESCO Weltkulturerbe seit 1997.)

```
windmills = mpimg.imread('windmills.png')

plt.axis("off")
plt.imshow(windmills)
plt.imshow(windmills)
```

```
<matplotlib.image.AxesImage at 0x7fd05c8f1190>
```

Wir möchten nun das Bild tönen. Das bedeutet wir "mischen" unsere Farben mit weiss. Dies wird die Helligkeit des Bildes erhöhen. Wir schreiben dafür eine Python-Funktion, welche ein Bild und eine Prozent-Angabe als Parameter entgegen nimmt. "percentage" auf 0 zu setzen, wird das Bild nicht verändern. Wenn Sie es auf eins setzen, wird das Bild vollständig aufgehellt:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def tint(imag, percent):
    """
    imag: the image which will be shaded
    percent: a value between 0 (image will remain unchanged
             and 1 (image will completely white)
    """
```

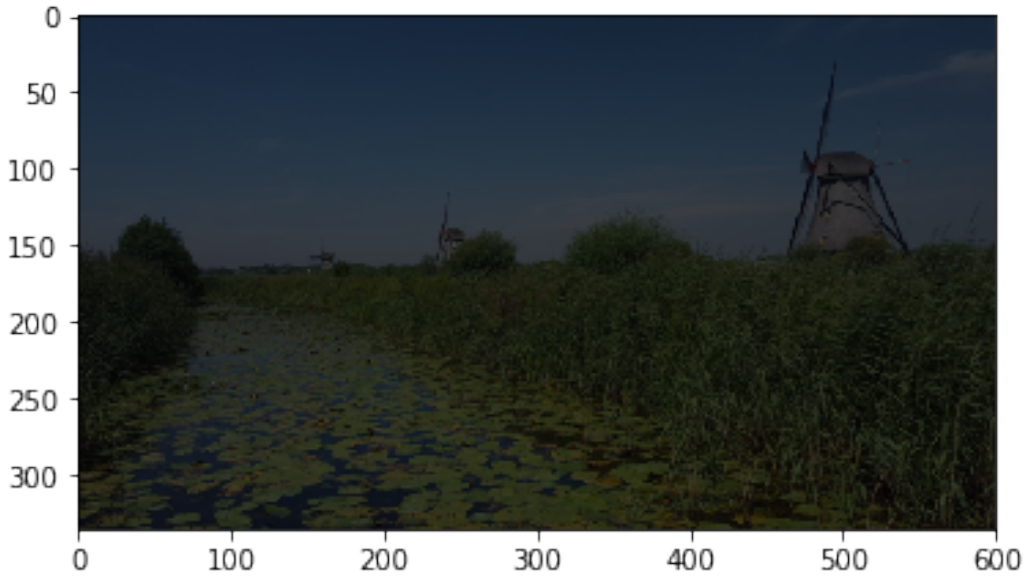
```
    tinted_imag = imag + (np.ones(imag.shape) - imag) * percent
    return tinted_imag

windmills = mpimg.imread('windmills.png')

tinted_windmills = tint(windmills, 0.8)
plt.axis("off")
plt.imshow(tinted_windmills)
plt.imshow(tinted_windmills)
```

```
<matplotlib.image.AxesImage at 0x7fd05e3dba90>
```


Ein Schatten ist eine Mischung einer Farbe mit Schwarz, was die Helligkeit verringert.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def shade(imag, percent):
    """
    imag: the image which will be shaded
    percent: a value between 0 (image will remain unchanged
             and 1 (image will be blackened)
    """
    tinted_imag = imag * (1 - percent)
    return tinted_imag

windmills = mpimg.imread('windmills.png')

tinted_windmills = shade(windmills, 0.7)
plt.imshow(tinted_windmills)
```

```
<matplotlib.image.AxesImage at 0x7fd05e516310>
```




```
def vertical_gradient_line(image, reverse=False):
    """
    We create a horizontal gradient line with the shape (1, image.shape[1], 3))
    The values are incremented from 0 to 1, if reverse is False,
    otherwise the values are decremented from 1 to 0.
    """
    number_of_columns = image.shape[1]
    if reverse:
        C = np.linspace(1, 0, number_of_columns)
    else:
        C = np.linspace(0, 1, number_of_columns)
    C = np.dstack((C, C, C))
    return C

horizontal_brush = vertical_gradient_line(windmills)
tinted_windmills =  windmills * horizontal_brush
plt.axis("off")
plt.imshow(tinted_windmills)
```

```
<matplotlib.image.AxesImage at 0x7fd06077e090>
```

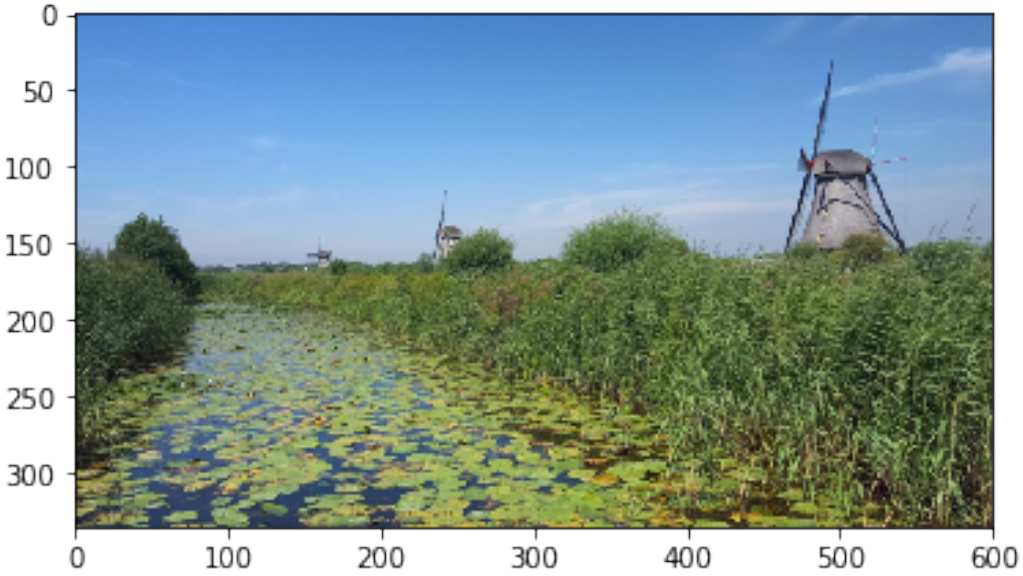
Wir werden das Bild von rechts nach links tönen indem wir den “reverse” Parameter der Python-Funktion auf “True” setzen:

```
def vertical_gradient_line(image, reverse=False):
    """
    We create a horizontal gradient line with the shape (1, image.shape[1], 3))
    The values are incremented from 0 to 1, if reverse is False,
    otherwise the values are decremented from 1 to 0.
    """
    number_of_columns = image.shape[1]
    if reverse:
        C = np.linspace(1, 0, number_of_columns)
    else:
        C = np.linspace(0, 1, number_of_columns)
    C = np.dstack((C, C, C))
    return C

horizontal_brush = vertical_gradient_line(windmills, reverse=True)
tinted_windmills =  windmills * horizontal_brush
plt.axis("off")
plt.imshow(tinted_windmills)
```

```
<matplotlib.image.AxesImage at 0x7fd05c852110>
```

```
def horizontal_gradient_line(image, reverse=False):
    """
    We create a vertical gradient line with the shape (image.shape[0], 1, 3))
    The values are incremented from 0 to 1, if reverse is False,
    otherwise the values are decremented from 1 to 0.
    """
    number_of_rows, number_of_columns = image.shape[:2]
    C = np.linspace(1, 0, number_of_rows)
    C = C[np.newaxis,:]
    C = np.concatenate((C, C, C)).transpose()
    C = C[:, np.newaxis]
    return C

vertical_brush = horizontal_gradient_line(windmills)
tinted_windmills =  windmills
plt.imshow(tinted_windmills)
```

```
<matplotlib.image.AxesImage at 0x7fd05c80bd10>
```

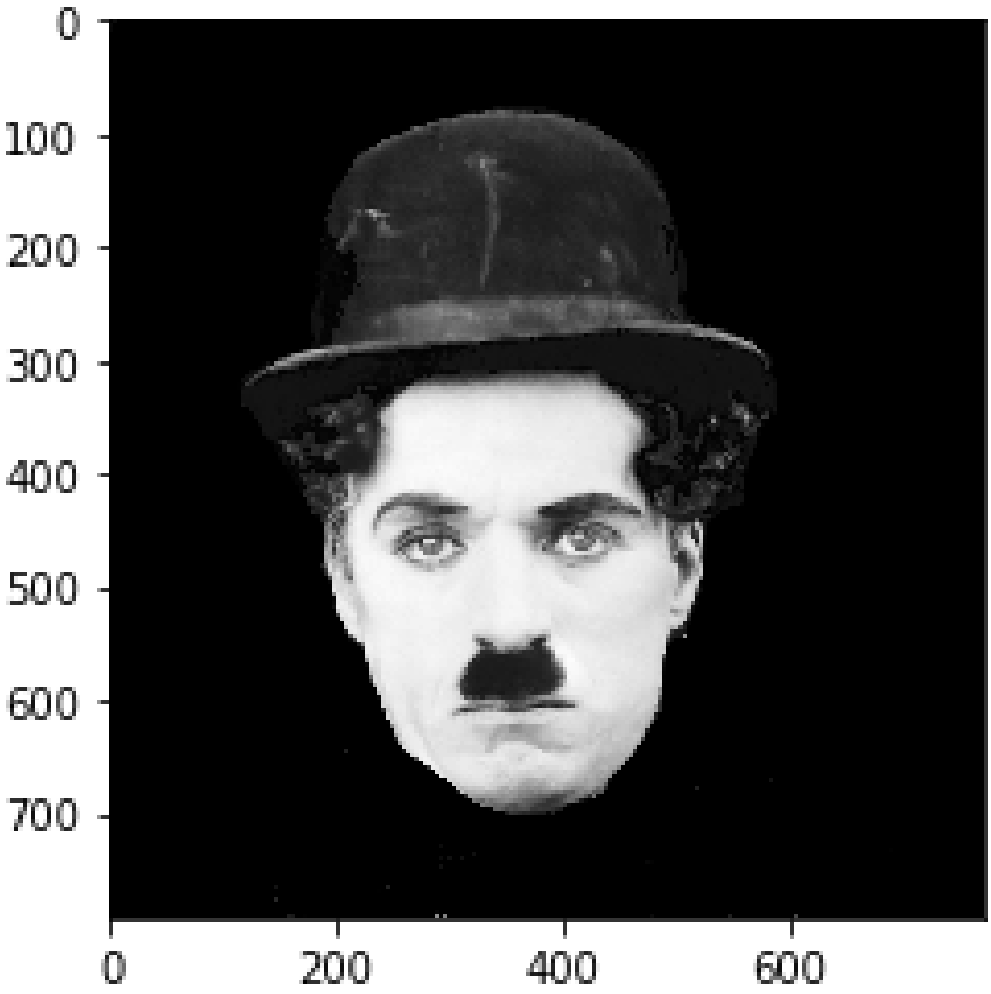


Ein Farbton wird entweder durch die Mischung einer Farbe mit Grau produziert, oder durch gleichzeitige Tönung und Schattierung.

```
charlie = mpimg.imread('Chaplin.png')
plt.gray()
print(charlie)
plt.imshow(charlie)
```

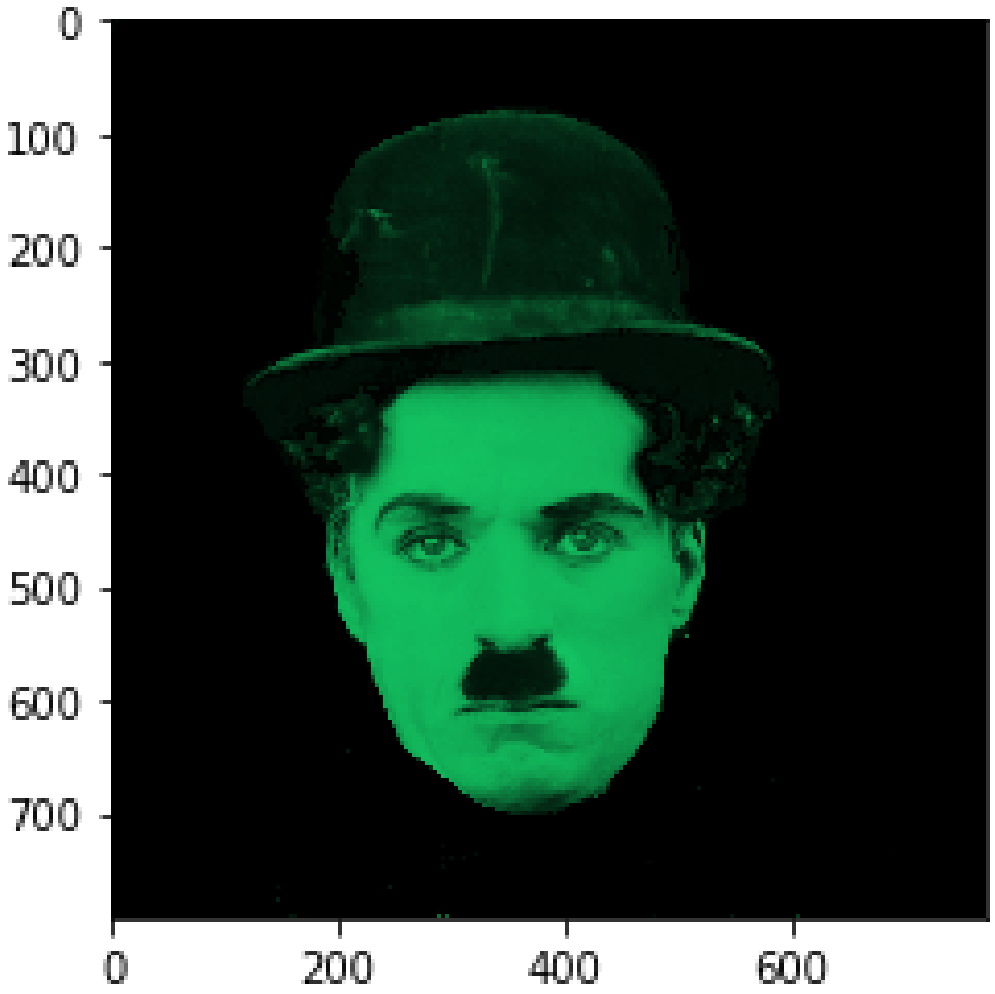
```
[[0.16470589 0.16862746 0.1764706  ... 0.         0.         0.        ]
 [0.16078432 0.16078432 0.16470589 ... 0.         0.         0.        ]
 [0.15686275 0.15686275 0.16078432 ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```

```
<matplotlib.image.AxesImage at 0x7fd05c774310>
```




```
colored = np.dstack((charlie*0.1, charlie*1, charlie*0.5))

plt.imshow(colored)
```

```
<matplotlib.image.AxesImage at 0x7fd05c6d6f50>
```



Im folgenden Beispiel werden verschiedene “colormaps” verwendet. Die “colormaps” befinden sich in `matplotlib.pyplot.cm.datad`:

```
plt.cm.datad.keys()
```

```
dict_keys(['Blues', 'BrBG', 'BuGn', 'BuPu', 'CMRmap', 'GnBu', 'Greens', 'Greys',
↪  'OrRd', 'Oranges', 'PRGn', 'PiYG', 'PuBu', 'PuBuGn', 'PuOr', 'PuRd',
↪  'Purples', 'RdBu', 'RdGy', 'RdPu', 'RdYlBu', 'RdYlGn', 'Reds', 'Spectral',
↪  'Wistia', 'YlGn', 'YlGnBu', 'YlOrBr', 'YlOrRd', 'afmhot', 'autumn', 'binary',
↪  'bone', 'brg', 'bwr', 'cool', 'coolwarm', 'copper', 'cubehelix', 'flag',
↪  'gist_earth', 'gist_gray', 'gist_heat', 'gist_ncar', 'gist_rainbow',
↪  'gist_stern', 'gist_yarg', 'gnuplot', 'gnuplot2', 'gray', 'hot', 'hsv',
↪  'jet', 'nipy_spectral', 'ocean', 'pink', 'prism', 'rainbow', 'seismic',
↪  'spring', 'summer', 'terrain', 'winter', 'Accent', 'Dark2', 'Paired',
↪  'Pastel1', 'Pastel2', 'Set1', 'Set2', 'Set3', 'tab10', 'tab20', 'tab20b',
↪  'tab20c', 'Blues_r', 'BrBG_r', 'BuGn_r', 'BuPu_r', 'CMRmap_r', 'GnBu_r',
↪  'Greens_r', 'Greys_r', 'OrRd_r', 'Oranges_r', 'PRGn_r', 'PiYG_r', 'PuBu_r',
↪  'PuBuGn_r', 'PuOr_r', 'PuRd_r', 'Purples_r', 'RdBu_r', 'RdGy_r', 'RdPu_r',
↪  'RdYlBu_r', 'RdYlGn_r', 'Reds_r', 'Spectral_r', 'Wistia_r', 'YlGn_r',
↪  'YlGnBu_r', 'YlOrBr_r', 'YlOrRd_r', 'afmhot_r', 'autumn_r', 'binary_r',
↪  'bone_r', 'brg_r', 'bwr_r', 'cool_r', 'coolwarm_r', 'copper_r',
↪  'cubehelix_r', 'flag_r', 'gist_earth_r', 'gist_gray_r', 'gist_heat_r',
↪  'gist_ncar_r', 'gist_rainbow_r', 'gist_stern_r', 'gist_yarg_r', 'gnuplot_r',
↪  'gnuplot2_r', 'gray_r', 'hot_r', 'hsv_r', 'jet_r', 'nipy_spectral_r',
↪  'ocean_r', 'pink_r', 'prism_r', 'rainbow_r', 'seismic_r', 'spring_r',
↪  'summer_r', 'terrain_r', 'winter_r', 'Accent_r', 'Dark2_r', 'Paired_r',
↪  'Pastel1_r', 'Pastel2_r', 'Set1_r', 'Set2_r', 'Set3_r', 'tab10_r', 'tab20_r',
↪  'tab20b_r', 'tab20c_r'])
```

```python
import numpy as np
import matplotlib.pyplot as plt

charlie = plt.imread('Chaplin.png')

#  colormaps plt.cm.datad
# cmaps = set(plt.cm.datad.keys())
cmaps = {'afmhot', 'autumn', 'bone', 'binary', 'bwr', 'brg',
         'CMRmap', 'cool', 'copper', 'cubehelix', 'Greens'}

X = [(4, 3, 1, (1, 0, 0)),
     (4, 3, 2, (0.5, 0.5, 0)),
     (4, 3, 3, (0, 1, 0)),
     (4, 3, 4, (0, 0.5, 0.5)),
     (4, 3, (5, 8), (0, 0, 1)),
     (4, 3, 6, (1, 1, 0)),
     (4, 3, 7, (0.5, 1, 0) ),
     (4, 3, 9, (0, 0.5, 0.5)),
     (4, 3, 10, (0, 0.5, 1)),
     (4, 3, 11, (0, 1, 1)),
     (4, 3, 12, (0.5, 1, 1))]

fig = plt.figure(figsize=(6, 5))

#fig.subplots_adjust(bottom=0, left=0, top = 0.975, right=1)
for nrows, ncols, plot_number, factor in X:
    sub = fig.add_subplot(nrows, ncols, plot_number)
    sub.set_xticks([])

    sub.imshow(charlie*0.0002, cmap=cmaps.pop())
    sub.set_yticks([])
```

2 Bildverarbeitungs Techniken

2.0.1 Techniken der Bildverarbeitung

Einführung

Sie werden vielleicht bemerkt haben, dass jede unsere Seiten der verschiedenen Tutorials mit einem Bild beginnen, welches erstellt wurde, um den Inhalt zu bereichern. Eines dieser Bilder ist das “raison d’être” dieses Kapitels. Wir möchten demonstrieren, wie wir das Bild für das Kapitel Dekorateure erstellt haben. Die Idee war mit Dekorateuren im “richtigen Leben” zu spielen. Kleine Icons mit Bildern von kleinen Arbeitern die einen Raum verzieren und auf der anderen Seite wird es mit einem “at”-Zeichen, dem Python-Symbol für Dekorateur, überblendet. Es ist somit ebenfalls ein gutes Beispiel wie man ein Wasserzeichen erstellt.

Wir erläutern in diesem Kapitel den gesamten Prozess, wie wir dieses Bild erstellt haben. Das Bild auf der rechten Seite wurde auf die gleiche Art und Weise erstellt. Hier wird allerdings ein Direktoren-Stuhl als Wasserzeichen verwendet, statt des “at”-Zeichens.

Zuerst schreiben wir eine Funktion “imag_tile” um das Bild sowohl horizontal, als auch vertikal zu kacheln. Dies verwenden dann als Hintergrund für unser Bild.

Anschliessend zeigen wir, wie man einen Ausschnitt oder einen Auszug per Slicing aus dem Bild holen kann. Wir werden die shade-Funktion verwenden, die wir im vorigen Kapitel über Bildverarbeitung erstellt haben, um das Bild zu schattieren.

Abschliessend werden wir aus dem Original-Bild, dem schattierten Bild und einem Bild mit dem “at”-Zeichen anhand der bedingten where-Funktion das finale Bild erstellen. Das finale Bild beinhaltet dann das “at”-Zeichen als Wasserzeichen, dass aus dem schattierten Bild ausgeschnitten ist.

Kachelung eines Bildes

Die Funktion “imag_tile”, die wir nun designen möchten, kann anhand des folgenden Diagramms am besten erklärt werden:

```
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import numpy as np

def imag_tile(img, n, m=1):
    """
    The image "img" will be repeated n times in
    vertical and m times in horizontal direction.
    """
```

```
    if n == 1:
        tiled_img = img
    else:
        lst_imgs = []
        for i in range(n):
             lst_imgs.append(img)
        tiled_img = np.concatenate(lst_imgs, axis=1 )
    if m > 1:
        lst_imgs = []
        for i in range(m):
             lst_imgs.append(tiled_img)
        tiled_img = np.concatenate(lst_imgs, axis=0 )

    return tiled_img

basic_pattern = mpimg.imread('decorators_b2.png')

decorators_img = imag_tile(basic_pattern, 3, 3)

plt.axis("off")
plt.imshow(decorators_img)
```

```
<matplotlib.image.AxesImage at 0x7f29cf529a20>
```

Ein Bild ist ein Drei-Dimensionales numpy ndarray.

```
type(basic_pattern)
```

```
numpy.ndarray
```

Die ersten drei Zeilen unseres Bildes basic_pattern sehen folgendermaßen aus:

```
basic_pattern[:3]
```

```
array([[[ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
```

```
        [ 1.,  1.,  1.],
        ...,
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.]],

       [[ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        ...,
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.]],

       [[ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        ...,
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.],
        [ 1.,  1.,  1.]]], dtype=float32)
```

Die innenliegendste Liste unseres Bildes beinhaltet die Pixel. Wir haben drei Werte die den R-, G- und B-Werten entsprechen. Wir haben also ein 24-bit PNG Bild. Für jeden RGB-Wert 8 bits.

PNG-Bilder können auch aus 32-bit Bildern bestehen (RGBA). Der vierte Wert "A" wird für die Transparenz benutzt (Ein-Kanal-Graustufen).

Es ist einfach auf individuelle Pixel anhand des Index zuzugreifen, z.B. in Zeile 100 und Spalte 20:

```
basic_pattern[100, 28]
```

```
array([ 0.90196079,  0.89019608,  0.86274511], dtype=float32)
```

Wie wir sehen, handelt es sich bei den Pixeln um Float-Werte (float32) zwischen 0 und 1. Matplotlib-Plotting kann sowohl mit float32 als auch unit8 für PNG-Bildern umgehen. Für alle anderen Bild-Formate wird nur unit8 unterstützt.

Bilder zuschneiden

Mit der Slicing-Funktion können Teil-Bilder ausgeschnitten werden. Wir schneiden im folgenden Beispiel das Bild von (90, 50), d.h. Zeile 90 und Spalte 50, bis (50, 120):

```
cropped = basic_pattern[90:150, 50:120]
plt.axis("off")
plt.imshow(cropped)
```

```
<matplotlib.image.AxesImage at 0x7f29cf3bdd30>
```

Wir brauchen diese Technik im folgenden.

Wir laden das Bild eines at-Zeichens im folgenden Skript:

Wir können die Slicing-Funktion um Teile aus einem Bild zu schneiden. Wir nutzen dies um sicherzustellen, dass beide Bilder die selbe Größe haben.

```
at_img=mpimg.imread('at_sign.png')

# at_img and decorators_img have to be of equal size:
d_shape = decorators_img.shape
at_shape = at_img.shape
height, width, colours = [min(x) for x in zip(*(d_shape, at_shape))]
at_img = at_img[0:height, 0:width]
```

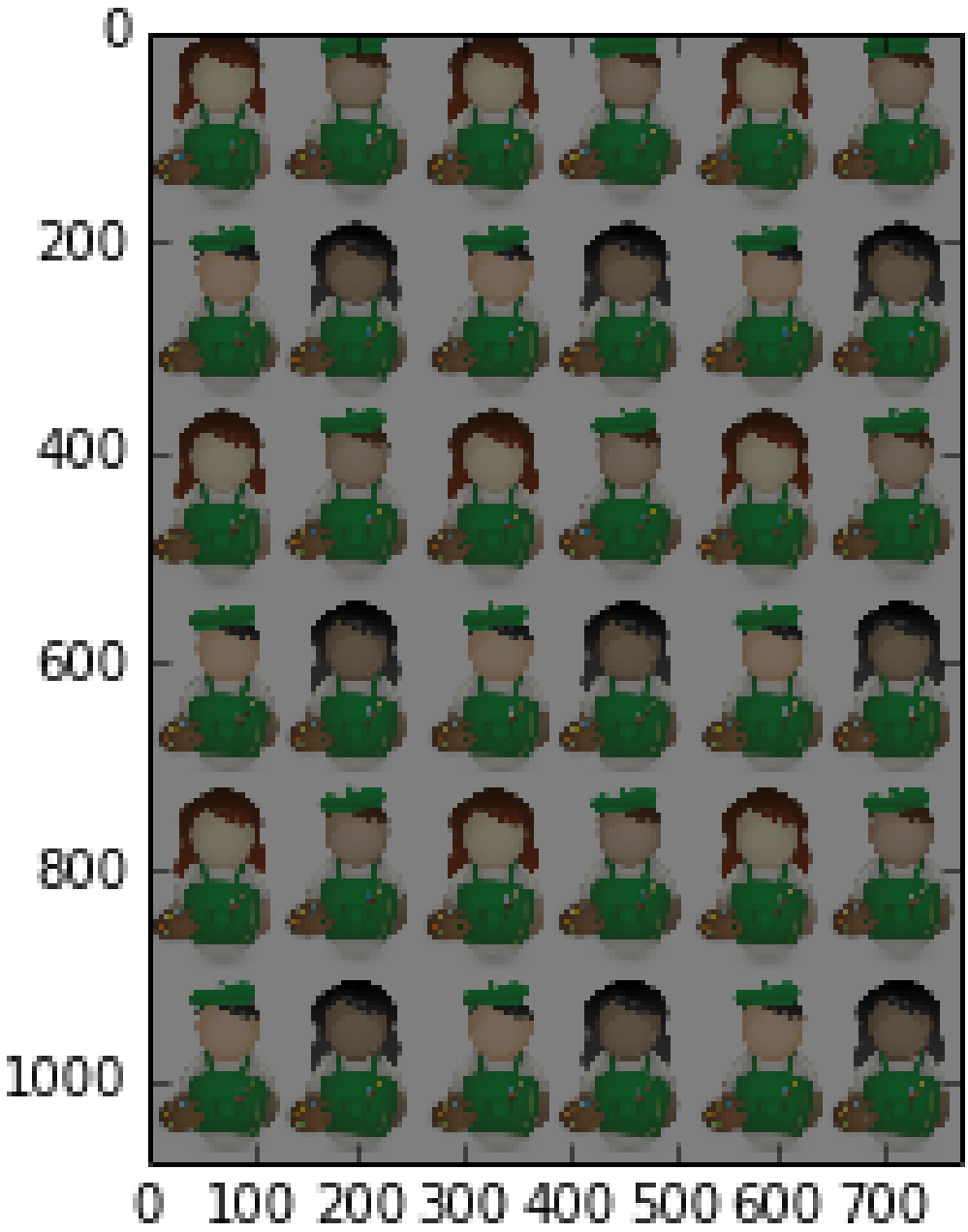
Schattieren eines Bildes

Wir definieren eine Funktion “shade” im folgenden Skript. “shade” erwartet zwei Parameter. Der erste Parameter “imag” ist das Bild, welches schattiert werden soll. Der zweite Parameter ist der Schattierungs-Faktor. Dies kann ein Wert zwischen 0 und 1 sein. Wenn der Faktor auf 0 gesetzt wird, so wird das Bild nicht verändern. Wenn der Faktor auf 1 gesetzt wird, wird das Bild komplett geschwärzt.

```
def shade(imag, percent):
    """
```

```
    imag: the image which will be shaded
    percent: a value between 0 (image will remain unchanged
             and 1 (image will be blackened)
    """
    tinted_imag = imag * (1 - percent)
    return tinted_imag

tinted_decorator_img = shade(decorators_img, 0.5)
plt.imshow(tinted_decorator_img)
print(tinted_decorator_img[:3])
```

```
[[[ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]
  ...,
  [ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]]

 [[ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]
  ...,
  [ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]]

 [[ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]
  ...,
  [ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]
  [ 0.5  0.5  0.5]]]
```



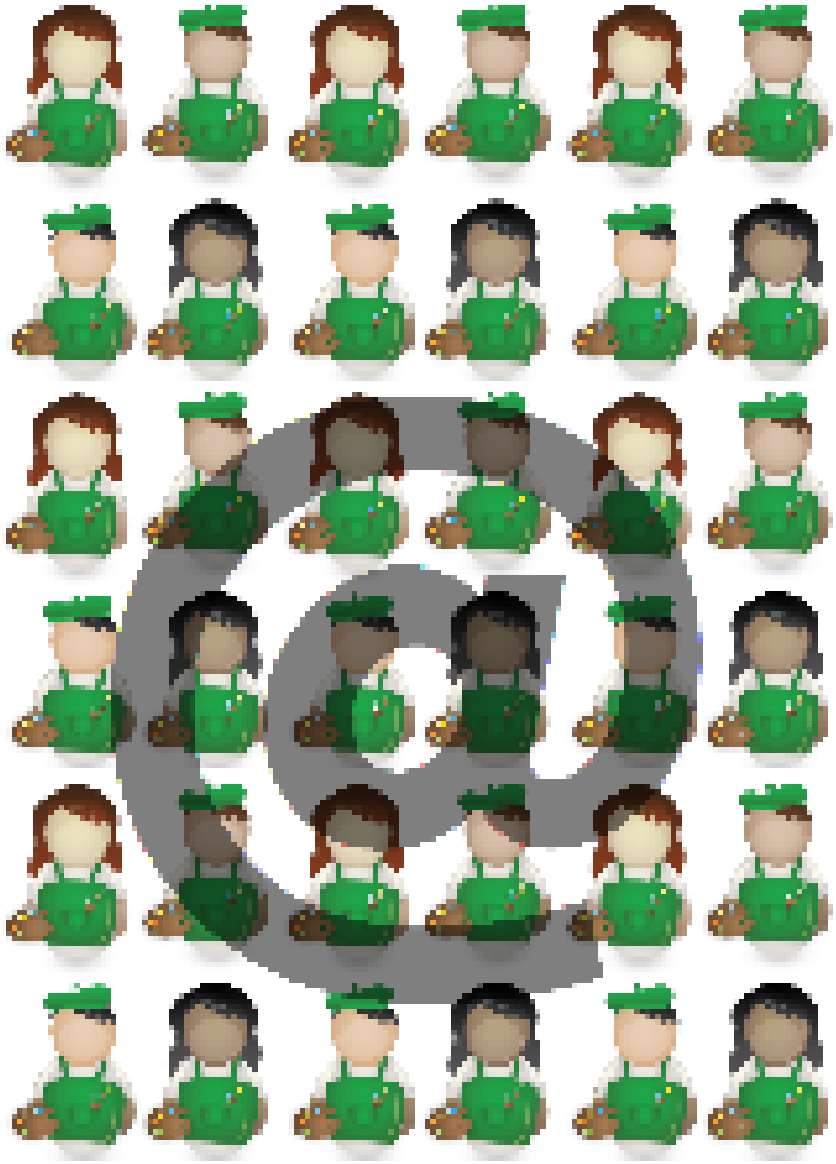
Überblendung von Bildern

Erstes Beispiel Wir haben nun alles zusammen um ein überblendetes Bild zu erstellen. Unser “at”-Zeichen besteht aus schwarzen und weißen Pixeln. Das überblendete Bild ist folgendermaßen aufgebaut: p=(n,m) sei ein beliebiges Pixel in der n-ten Zeile und der m-ten Spalte des at_img Bildes. Wenn der Wert des Pixels nicht schwarz oder dunkelgrau ist, verwenden wir das Pixel an der Position (n,m) des Bildes decorators_img. Andernfalls verwenden wir das entsprechende Pixel aus tinted_decorator_img. Die where-Funktion von numpy ist ideal für diese Aufgabe:

```
print(at_img.shape,
      decorators_img.shape,
```

```
      tinted_decorator_img.shape)
basic_pattern = mpimg.imread('decorators2.png')
img2 = np.where(at_img > [0.1, 0.1, 0.1],
                decorators_img,
                tinted_decorator_img)

plt.axis("off")
plt.imshow(img2)
```

```
(1077, 771, 3) (1077, 771, 3) (1077, 771, 3)

<matplotlib.image.AxesImage at 0x7f29cf422b70>
```



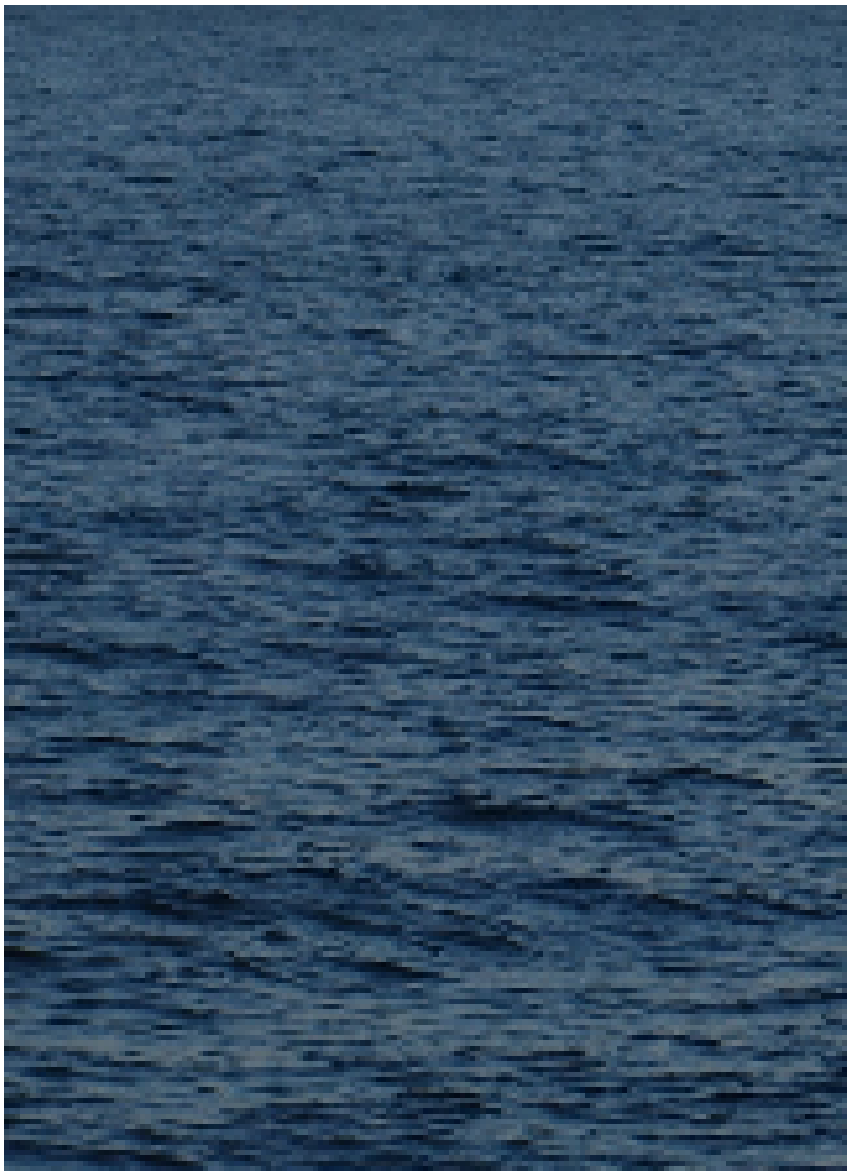
Was noch zu tun ist, ist das neu erstellte Bild zu speichern:

```
mpimg.imsave('decorators_with_at.png', img2)
```

Zweites Beispiel Jetzt wollen wir ein anderes Bild als "Wasserzeichen" verwenden. Anstatt des "at"-Zeichens, verwenden wir jetzt ein Direktoren-Stuhl. Wir erstellen jetzt das Bild, dass wir zu Beginn genannt haben.

```
images = [mpimg.imread(fname) for fname in ["director_chair.png",
↪  "the_sea.png", "the_sky.png"]]
director_chair, sea, sky = images


plt.axis("off")
plt.imshow(sea)
```

```
<matplotlib.image.AxesImage at 0x7f29cf4274a8>
```



```
plt.axis("off")
plt.imshow(director_chair)
```

Im folgenden, überblenden wir die Bilder director_chair, decorators_img und sea indem wir nochmal die where-Funktion verwenden:

```
#sea2 = mpimg.imread('the_sea2.png')
img = np.where(director_chair > [0.9, 0.9, 0.9],
               decorators_img,
               sea)
plt.axis("off")
plt.imshow(img)
mpimg.imsave('decorators_with_chair', img)
```

Wir könnten statt mpimg.imread aus matplotlib auch Image.open aus PIL verwenden um die Bilder zu lesen. Es gibt einen entscheidenden Unterschied oder ein potentielles "Problem" zwischen den beiden Varianten: Das Bild das von imread gelesen wird, hat Werte zwischen 0 und 1. Dagegen hat liefert Image.open Werte zwischen 0 und 255. Somit müssen wir alle Werte durch 255 dividieren, wenn wir mit einem Bild arbeiten müssen, dass mit mpimg.imread gelesen wurde:

```
from PIL import Image


img = Image.open("director_chair.jpg")
img = img.resize((at_img.shape[1], at_img.shape[0]))
img = np.asarray(img)
```

```
plt.axis("off")
plt.imshow(img)

print(img[100, 129])
```

```
[27 27 27]
```

```
# PIL: Pixel are within range 0 and 255
# mpimg: range 0 bis 1

img = np.asarray(img, np.float)
```

```
img = img / 255

print(img[100, 129])
```

```
[ 0.00041522  0.00041522  0.00041522]
```

3 Video Aus Bildern Erzeugen

3.0.1 Videos aus Bildern erstellen

Dieses Kapitel unseres Python-Kurses beschäftigt sich mit Filmen und Bildern. Sie werden lernen wie Sie aus Bildern Filme erstellen können.

Ein Video aus mehreren Bildern

Sie haben also eine Menge Bilder und möchten daraus ein Video erstellen. Und zwar so, dass jedes Bild eine bestimmte Zeit lang gezeigt wird. Wir nehmen an, dass alle Ihre Bilder, - das können Fotos sein, die Sie in Ihrem letzten Urlaub gemacht haben oder Gemälde, die Sie zu einem bestimmten Zeitpunkt gemalt haben - in einem Verzeichnis sind. Wir gehen nun davon aus, dass alle Ihre Bilder die gleiche Größe haben, falls nicht, werden Sie auch lernen, wie Sie die Größe dieser Bilder ändern können. Für unsere Zwecke könnte es gut sein, dass die Bildnamen einem speziellen Benennungsschema folgen, d.h. sie beginnen alle mit dem gleichen Präfix (wie z.B. 'pic_') gefolgt von einer Zahl (z.B. '001', '002', '003', ...) in aufsteigender Reihenfolge. Der letzte Teil des Namens ist natürlich das Suffix, das dem eigentlichen Bildformat entsprechen muss.

Sie können alle benötigten Bilder und die erstellten Videos für den persönlichen Gebrauch herunterladen (das Copyright liegt bei Bernd Klein) unter python-courses: Material

Zunächst stellen wir eine Funktion 'seq_renaming' bereit, die alle Bilder in einem Ordner umbenennt nach dem zuvor beschriebenen Namensschema umbenennt:

```
import glob
import os

def seq_renaming(folder='.', prefix='pic_', start=0, num_length=3):
    count = start
    folder += '/'
    for fname in glob.glob(folder + '*.jpg'):
        suffix = fname[fname.rfind('.'):]
        outstr = f"{folder}/{prefix}{count:0{num_length}d}{suffix}"
        count += 1
        os.rename(fname, outstr)

seq_renaming('flower_images')
```

Um unsere neu erstellte Funktion zu testen und zu demonstrieren, wie sie funktioniert, erstellen wir ein Testverzeichnis mit leeren Dateien, die Bilder simulieren:

```
import shutil
target_dir = 'test_dir'

if os.path.exists(target_dir):
```

```
    # delete the existing directory
    shutil.rmtree(target_dir)
# Create target_dir, because it doesn't exist so far
os.makedirs(target_dir)

names = ['apples', 'oranges', 'bananas', 'pears']
for name in names:
    open(f'{target_dir}/{name}.jpg', 'w').write(' ')

print('Before renaming: ', os.listdir(target_dir))

# Let's rename the image names:
seq_renaming(target_dir)
print('After renaming: ', os.listdir(target_dir))
```

```
Before renaming:  ['apples.jpg', 'oranges.jpg', 'bananas.jpg', 'pears.jpg']
After renaming:  ['pic_000.jpg', 'pic_001.jpg', 'pic_002.jpg', 'pic_003.jpg']
```

Unser erstes Video wird mit Hilfe der Bilder im Ordner flower_images erstellt. Schauen wir uns eines der Bilder an.

```
import matplotlib.pyplot as plt
import random
import numpy as np
import cv2

image = plt.imread("im2video/images/20210617_201910.jpg")
plt.imshow(image)
#cv2.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7fa6f5bd8210>
```

Das Bild ist in Radolfzell am Bodensee aufgenommen und im Hintergrund sind die Berge Hohentwiel und Hohenstoffeln zu sehen.

```
import matplotlib.pyplot as plt
import random
```

```
import numpy as np
import os
import cv2
import glob

def get_frame_size(image_path):
    """ Reads an image and calculates
    the width and length of the images,
    which will be returned """
    frame = cv2.imread(image_path)
    height, width, layers = frame.shape
    frame_size = (width, height)
    return frame_size


def video_from_images(folder='.',
                      video_name = 'video.avi',
                      suffix='png',
                      prefix='pic_',
                      reverse=False,
                      length_of_video_in_seconds=None,
                      codec = cv2.VideoWriter_fourcc(*'DIVX')):
    """ Die Funktion erstellt ein Video aus allen Bildern mit
    dem Suffix 'suffix' (Standard ist 'png') und dem Präfix
    'prefix'(Standard ist 'pic_') im Ordner 'folder'.
    Wenn 'length_of_video_in_seconds' auf None gesetzt ist,
    ist dies die Anzahl der Bilder in Sekunden.
    Wenn ein positiver  Wert angegeben wird, handelt es sich
    um die Länge des Videos in Sekunden. Die Funktion geht davon
    aus, dass die 'shape' des ersten Bildes gleich für alle
    anderen Bilder ist. Ist dies nicht der Fall, so wird eine
    Warnung ausgegeben und die Größe wird entsprechend angepasst! """

    images = []
    for fname in glob.glob(f'{folder}/{prefix}*{suffix}'):
        images.append(fname)
    images.sort(reverse=reverse)
    if length_of_video_in_seconds is None:
        # jedes Bild wird für eine Sekunde angezeigt
        length_of_video_in_seconds = len(images)

    # Es wird die Anzahl der Frames pro Sekunde berechnet:
    frames_per_second = len(images) / length_of_video_in_seconds
    frame_size = get_frame_size(images[0])

    video = cv2.VideoWriter(video_name,
                            codec,
                            frames_per_second,
                            frame_size)

    for image in images:
        im = cv2.imread(image)
        height, width, layers = im.shape
        if (width, height) != frame_size:
            print(f'Warning: {image} resized from {(width,
            ↪  height)} to {frame_size}')
            im = cv2.resize(im, frame_size)
```

```
            video.write(im)
    cv2.destroyAllWindows()
    video.release()

video_from_images('im2video/flower_images', 'flowers.avi', 'jpg')
```

```
Warning: im2video/flower_images/pic_004.jpg resized from (3993, 1860) to (4000,
↪  1800)
Warning: im2video/flower_images/pic_005.jpg resized from (4000, 2337) to (4000,
↪  1800)
```

Das Ergebnis ist in flowers.avi zu sehen!

Ein Video aus einem Bild

Wir verwenden nun ein Bild als Grundlage, um eine Bildfolge zu erstellen. Wir tun dies, indem wir die Farben eines Teils (einer Tile / Kachel) des Bildes ändern. Wir können uns das Bild als ein Schachbrett mit n Zeilen und m Spalten vorstellen. Die Funktion random_tile kann verwendet werden, um die obere linke und untere rechte Ecke einer zufällig ausgewählten Kachel zu liefern. Wir übergeben das Bild an den Parameter 'img' und die Anzahl der Zeilen und Spalten werden durch height_factor, width_factor definiert:

```
import random

def random_tile(img, height_factor, width_factor):
    """ returns the top left and bottom right corner
    of a random tile in an image. The tile sizes are
    determined by height_factor, width_factor, the division
    factors """
    height, width, colours = img.shape
    tiles = height_factor, width_factor
    tile_height, tile_width = height / height_factor, width / width_factor
    tile_upper_left_row = int(round(random.randint(0, height_factor) *
    ↪  tile_height, 0))
    tile_upper_left_column = int(round(random.randint(0, width_factor) *
    ↪  tile_width, 0))
    top_left = tile_upper_left_row, tile_upper_left_column
    bottom_right_row = int(round(tile_upper_left_row + tile_height, 0))
    bottom_right_column = int(round(tile_upper_left_column + tile_width, 0))
    bottom_right = bottom_right_row, bottom_right_column
    return top_left, bottom_right
```

Im folgenden Code lesen wir ein Bild ein und rufen die Funktion random_tile einige Male auf:

```
import matplotlib.pyplot as plt
import random
import numpy as np
imag = cv2.imread("im2video/images/20210617_201910.jpg")
print(f'{imag.shape=}')
for i in range(4):
    tile = random_tile(imag, 4, 5)
    print(tile)
```

```
imag.shape=(873, 1941, 3)
((0, 1553), (218, 1941))
((655, 1941), (873, 2329))
((218, 388), (436, 776))
((436, 1553), (654, 1941))
```

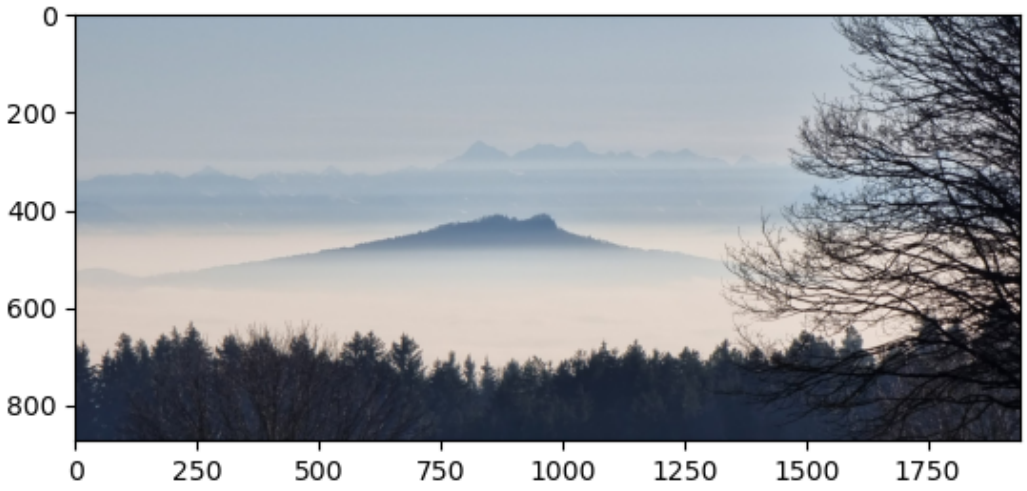
Die Funktion 'colorize_tile' kann verwendet werden, um die Farbe einer Kachel zufällig zu ändern. Dies geschieht durch Hinzufügen eines zufälligen Farbwert zu jedem Pixel der Kachel hinzugefügt wird.

```
def colorize_tile(imag, top_left, bottom_right, max_col_val):
    color = np.random.randint(0, max_col_val, (3,))
    r1, r2 = top_left[0], bottom_right[0]
    c1, c2 = top_left[1], bottom_right[1]
    imag[r1:r2, c1:c2] = imag[r1:r2, c1:c2] + color
```

Wir demonstrieren die Arbeitsweise von colorize_tile im folgenden Code. Schauen wir uns zunächst die Bilder an. Wir verwenden plt.imread anstelle von cv2.imread, weil cv2.imshow ein externes Fenster im Jupyter-Notebook öffnet, das wir zur Erstellung dieser Website verwenden. Außerdem sind die Bilder von cv2.imread im BGR-Format, während plt.imread RGB-Bilder liefert. Das bedeutet, dass sie nicht kompatibel sind.

```
import matplotlib.pyplot as plt
import random
import numpy as np
imag = plt.imread("im2video/images/20230215_161843.jpg")
plt.imshow(imag)
```

```
<matplotlib.image.AxesImage at 0x7fa6f5147190>
```

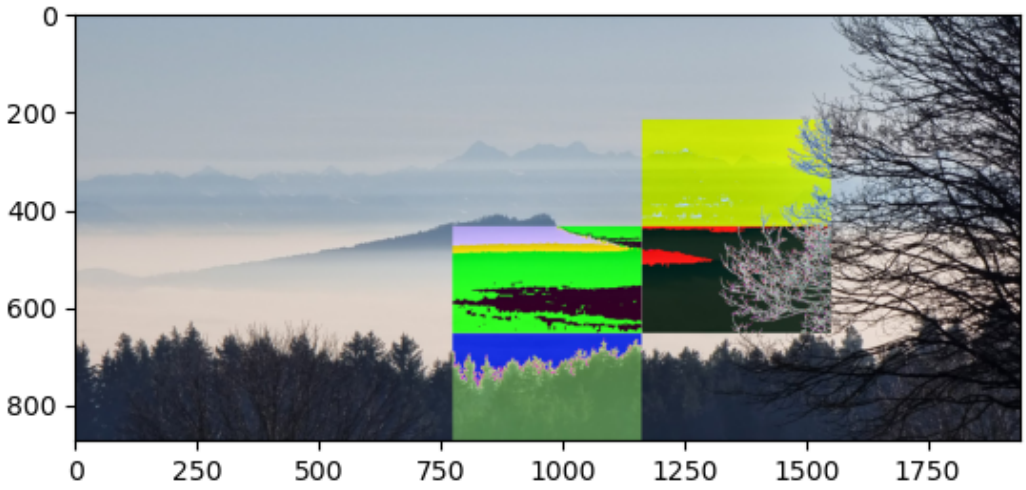


Lassen Sie uns jetzt mit colorize_tile experimentieren:

```
imag = cv2.imread("im2video/images/20230215_161843.jpg")
for i in range(6):
    top_left, bottom_right = random_tile(imag, 4, 5)
    colorize_tile(imag, top_left, bottom_right, max_col_val=100)
```

```
# Wir speichern das Ergebnis in 'experiments.png'
cv2.imwrite('experiments.png', imag)

# Wegen Einschränkungen des Jupyter-Noebook
# müssen wir plt zum Lesen und Schreiben verwenden:
plt.imshow(plt.imread('experiments.png'))
```

```
<matplotlib.image.AxesImage at 0x7fa6f5340f50>
```



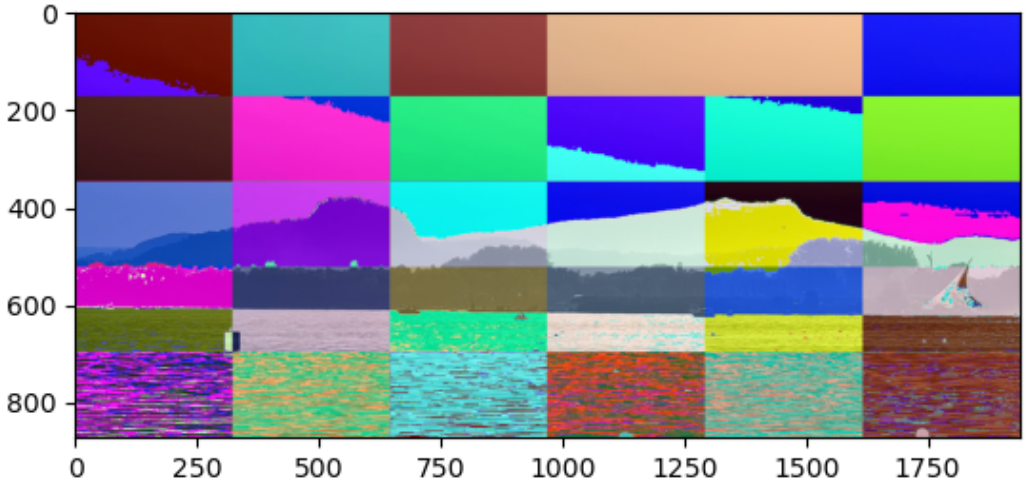
Wir werden nun 100 Bilder in einem Ordner mit dem Namen “video_pics” erstellen. Das Originalbild wird unter dem Namen pic_101.png in diesem Ordner gespeichert. Das Bild mit dem Namen “pic_001.png” ist das Bild mit den meisten geänderten Kacheln:

```
pics_dir='im2video/video_pics'
n = 100
im = cv2.imread("im2video/images/20210617_201910.jpg")
fname = f"{pics_dir}/pic_{n+1:03d}.png"
cv2.imwrite(fname, im)
for i in range(n, 0, -1):
    top_left, bottom_right = random_tile(im, 5, 6)
    colorize_tile(im, top_left, bottom_right, 100)
    fname = f"{pics_dir}/pic_{i:03d}.png"
    cv2.imwrite(fname, im)
```

Schauen wir uns das endgültige Bild “pic_001.png” der erstellten Bilder an:

```
plt.imshow(plt.imread(f'{pics_dir}/pic_001.png'))
```

```
<matplotlib.image.AxesImage at 0x7fa6f5237190>
```



Wir werden nun wieder unsere Funktion video_from_images verwenden, um aus diesen neu erstellten Bildern ein Video mit dem Namen 'rand_squares.avi' zu erstellen:

```
video_from_images(pics_dir,
                  'random_squares.avi',
                  'png',
                  length_of_video_in_seconds = 280)
```

Ein Video mit Kacheln (Tiles) aus mehreren Bildern

Wir werden nun Bilder erstellen, wobei wir ein Startbild verwenden und zufällig Kacheln durch die entsprechenden Kacheln anderer Bilder ersetzen. Die Funktion 'tile_from_image' ersetzt den Inhalt einer Kachel eines Bildes1 durch den Inhalt der entsprechenden Kachel eines anderen Bildes 'image2'. Beide Bilder müssen die gleiche Form haben:

```
def tile_from_image(image1, image2, top_left, bottom_right):
    """ Nimmt eine Kachel aus Bild2 und ersetzen den
    entsprechenden Bereich in Bild1 """
    r1, r2 = top_left[0], bottom_right[0]
    c1, c2 = top_left[1], bottom_right[1]
    image1[r1:r2, c1:c2] = image2[r1:r2, c1:c2]
```

Im folgenden Code verwenden wir Bilder, die aus der Bodensee-Region stammen:

```
images_dir = 'im2video/bodensee_area'
target_dir = 'video_pics_bodensee'
if not os.path.exists(target_dir):
   # target_dir erstellen, da es bisher nicht existiert
   os.makedirs(target_dir)
n_of_images = 120
import random

images = []
for fname in glob.glob(f'{images_dir}/*.jpg'):
    images.append(fname)
```

```
imags = [cv2.imread(fname) for fname in images]
start_image = imags[0]
for i in range(n_of_images, 0, -1):
    top_left, bottom_right = random_tile(start_image, 5, 6)
    image2 = random.choice(imags)
    tile_from_image(start_image, image2, top_left, bottom_right)
    fname = f"{target_dir}/pic_{i:03d}.png"
    cv2.imwrite(fname, start_image)
```

```
video_from_images('video_pics_bodensee',
                  'bodensee.avi',
                  'png',
                  length_of_video_in_seconds = 60)
```

Ich habe diese Technik verwendet, um ein Video für eine Musikkomposition von mir zu erstellen. Das Video ist bei YouTube zu finden:

- Bernd Klein - Out of the Mud, Version 3 oder
- Bernd Klein - Raus aus dem Schlamm, Version 2

Die Musik ist in beiden Videos die gleiche. Ein Musikstück für Streichorchester und Solocello. Die Tonspur habe ich mit dem Linux-Programm kdenlive hinzugefügt.

4 Videos Bewegliche Objekte

4.0.1 Videos mit beweglichen Objekten

Die Idee hinter diesem Kapitel unserer Python-Kurse ist es, einen Film aus einem oder mehreren Bildern zu erstellen, in dem ein oder mehrere Objekte auf statischen Bildern herumbewegt werden. Wir werden feste Objekte oder durchscheinende Wasserzeichenobjekte über Bilder bewegen. Wir haben ein Bild mit einer Wasserzeichenstruktur für unser Kapitel über Dekoratoren und Dekoration erstellt. Sie können es ganz oben auf der Seite sehen: Ein "kaufmännisches Und" (in der Python-Gemeinschaft besser bekannt als das Dekorator-Zeichen. Sie können auch einen Kurs finden, wie man Bilder mit Wasserzeichen wie dieses erstellt mit Python, Numpy, Scipy und Matplotlib in dem Kapitel Bildverarbeitungstechniken. Darin wird der gesamte Prozess der Erstellung unseres Dekorators und des Bildes mit dem Zeichen, d.h. der mit Wasserzeichen versehenen Bilder, erklärt. Es ist also eine gute Idee, diese Kapitel zu lesen, bevor man weitermacht.

Technische Anmerkung: opencv muss für die folgenden Python-Code-Beispiele installiert sein.

Interessante Objekte mit Matplotlib erstellen

Wir beginnen mit der Erstellung einiger interessanter Objekte, die wir in den Videos, die wir erstellen wollen, als bewegliche Objekte verwenden können. Wir beginnen mit der Erstellung eines Verzeichnisses, in dem die Objekte gespeichert werden:

```
import os
import shutil
target_dir = 'images4video'
watermarks_dir = f"{target_dir}/watermarks_dir"
if os.path.exists(watermarks_dir):
    # delete the existing directory
    shutil.rmtree(watermarks_dir)
# Create target_dir, because it doesn't exist so far
os.makedirs(watermarks_dir)
```

Wir verwenden im folgenden Code den Parameter `bbox_inches=ttight"` in `savefig`. Wenn dieser Parameter auf "tight" gesetzt ist, versucht matplotlib, alle zusätzlichen Leerzeichen oder Padding an den Rändern der Abbildung zu entfernen, so dass nur der eigentliche Inhalt der Abbildung gespeichert wird. Dies kann nützlich sein, wenn man eine Abbildung mit minimalen Rändern oder Auffüllungen speichern will.

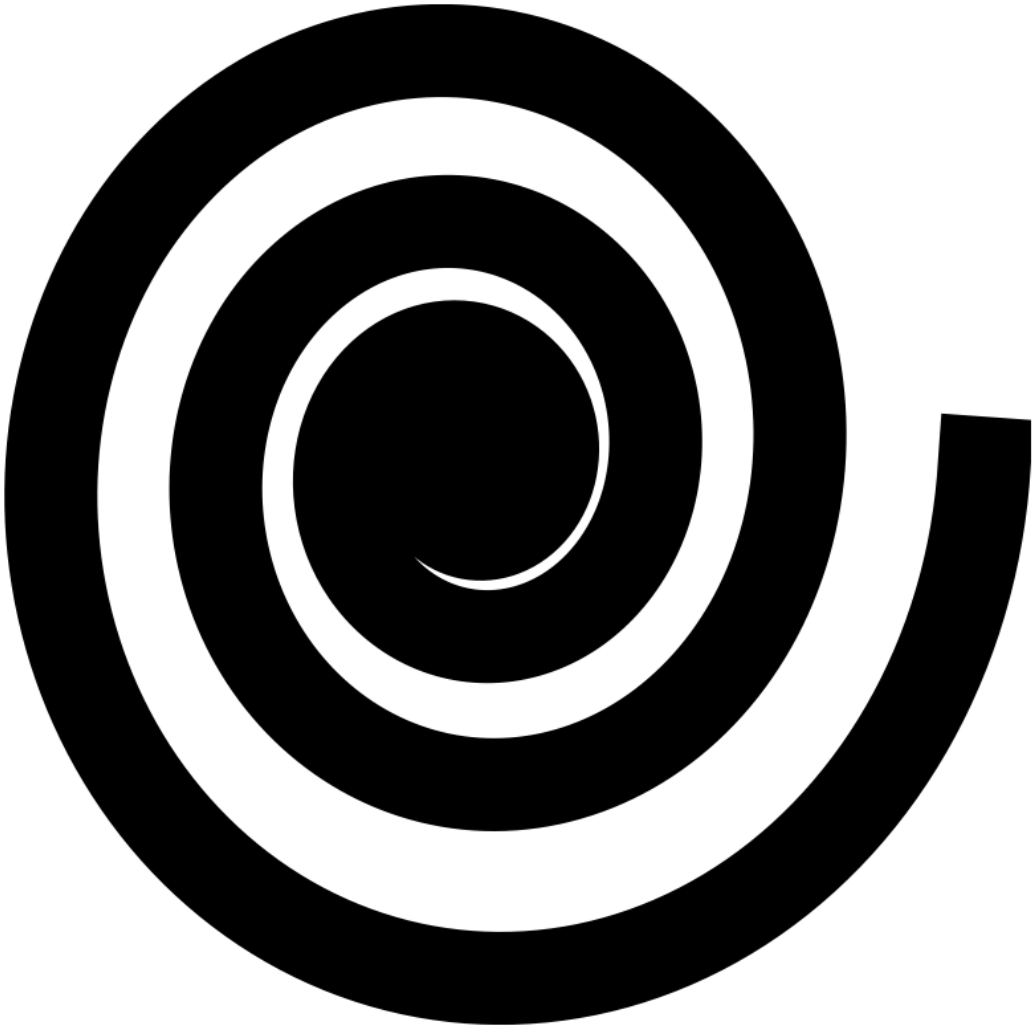
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 2, 1000)
y1 = np.sqrt(1-(abs(x)-1)**2)
```

```
y2 = -3 * np.sqrt(1-(abs(x)/2)**0.5)

fig, ax = plt.subplots()

ax.fill_between(x, y1, color='red')
ax.fill_between(x, y2, color='red')
ax.axis(xmin=-2.3, xmax=2.3)
ax.axis('off')
#ax.xlim([-2.5, 2.5])
txt2include = "python-kurs.eu"
ax.text(0, -0.4,
         txt2include,
         fontsize=24,
         fontweight='bold',
         color='orange',
         horizontalalignment='center')

plt.savefig(f"{watermarks_dir}/heart.png", bbox_inches="tight")
```

Wir erzeugen nun ein Bild mit einer schwarz-weißen Spirale:

```
import numpy as np
import matplotlib.pyplot as plt

theta = np.radians(np.linspace(0, 360*5, 1000))
r = theta**2
x_2 = r*np.cos(theta)
y_2 = r*np.sin(theta)
plt.figure(figsize=[5, 5])
```

```
plt.gcf().set_size_inches(10, 10)
plt.plot(x_2, y_2, linewidth=50.5, color='black')
plt.axis('off')
plt.savefig(f"{watermarks_dir}/spiral.png", bbox_inches="tight")
plt.show()
```



Wie sieht es mit Farben aus? Eine andere Spirale, aber jetzt in Farben:

```
import matplotlib.pyplot as plt
import numpy as np

theta = np.arange(0, 8*np.pi, 0.1)
a, b = 1, 0.5

for dt in np.arange(0, 2*np.pi, np.pi/2.0):
    x = a*np.cos(theta + dt)*np.exp(b*theta)
    y = a*np.sin(theta + dt)*np.exp(b*theta)
    dt = dt + np.pi/4.0
    x2 = a*np.cos(theta + dt)*np.exp(b*theta)
    y2 = a*np.sin(theta + dt)*np.exp(b*theta)
    xf = np.concatenate((x, x2[::-1]))
    yf = np.concatenate((y, y2[::-1]))
```

```
    p1 = plt.fill(xf, yf)

plt.axis('equal')
plt.axis('off')
plt.tight_layout()
plt.savefig(f'{watermarks_dir}/coloured_spiral.png', bbox_inches="tight")
```

Größe eines Bildes

Die folgende Funktion gibt die Größe eines Bildes als Tupel (Breite, Höhe) zurück. opencv muss installiert sein, damit Sie das cv2-Modul verwenden können:

```
import cv2

def get_frame_size(image_path):
    """ Reads an image and calculates
    the width and length of the images,
    which will be returned """
    frame = cv2.imread(image_path)
    height, width, layers = frame.shape
    frame_size = (width, height)
    return frame_size

get_frame_size(f"{watermarks_dir}/spiral.png")
```

```
(794, 790)
```

4.0.2 Zufällige Kachel (tile) im Bild

Jetzt definieren wir eine Funktion `random_tile`, die die Koordinaten (obere linke Ecke und untere rechte Ecke) einer Kachel im Bild `img` zurückgibt. Die Größe dieser Kachel ist gleich der Größe des Objektes bzw. Wasserzeichenbildes, das der Funktion übergeben wird:

```
import random

def random_tile(img, watermark_image):
    """ returns the top left and bottom right corner
    of a random tile in an image. The size correspondents to
    the size of the watermake image """
    height, width, colours = img.shape
    w_height, w_width, colours = watermark_image.shape
    tile_upper_left_row = int(round(random.randint(0, height - w_height), 0))
    tile_upper_left_column = int(round(random.randint(0, width - w_width), 0))
    top_left = tile_upper_left_row, tile_upper_left_column
    bottom_right_row = int(round(tile_upper_left_row + w_height, 0))
    bottom_right_column = int(round(tile_upper_left_column + w_width, 0))
    bottom_right = bottom_right_row, bottom_right_column
    return top_left, bottom_right
```

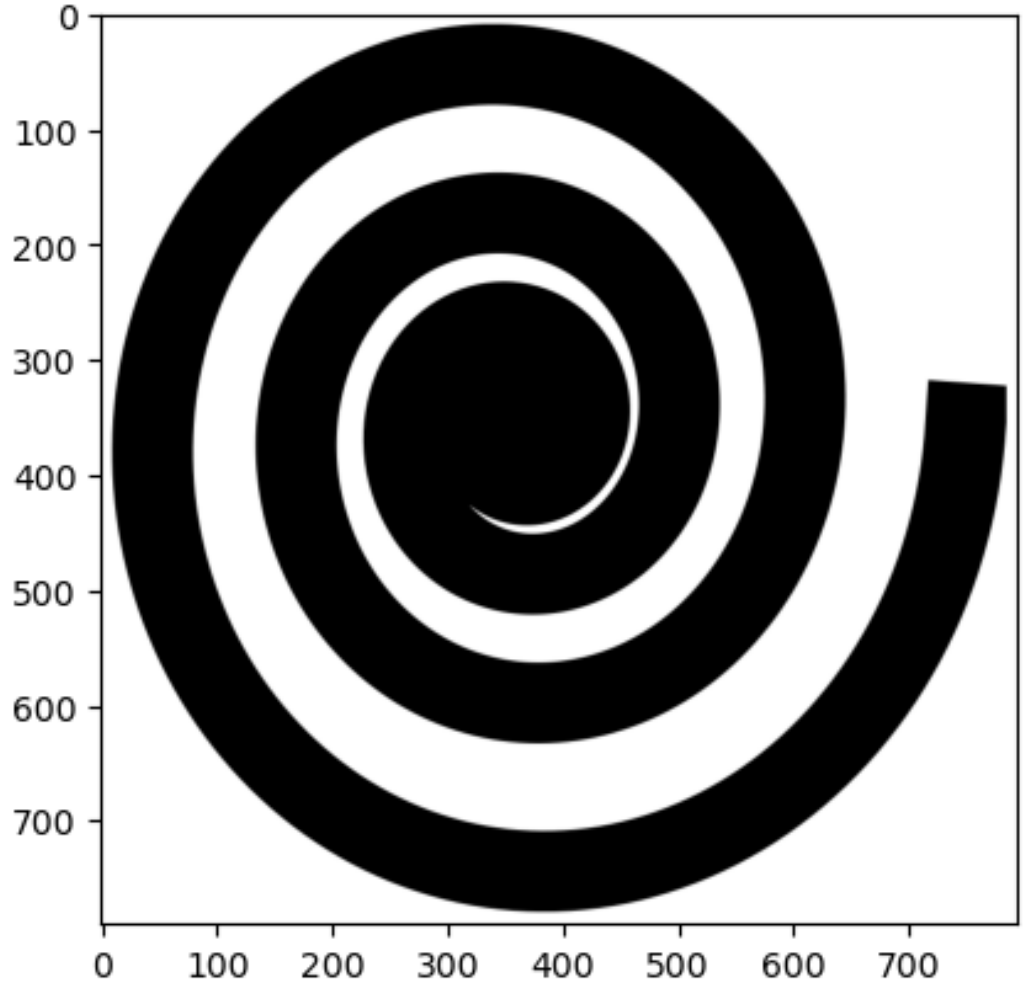
Wir werden cv2.imread verwenden, um ein Bild einzulesen, das wir in unseren Beispielen verwenden wollen. Wir könnten auch cv2.imshow verwenden, aber das öffnet ein externes Fenster im Jupyter-Notebook, das wir zum Erstellen dieser Website verwenden. Das ist der Grund, warum wir plt.imshow verwenden, um das Bild zu betrachten. Wir müssen die Farben spiegeln, bevor wir plt.imshow verwenden können, da wir mit cv2.imread ein BGR-Bild (blau, grün, rot) erhalten, während plt.imread RGB-Bilder liefert.

```
import matplotlib.pyplot as plt
import random
import numpy as np
import cv2

image = cv2.imread("images4video/paths/pic_004.jpg")
rgb_img = image[:, :, ::-1]  # BGR ---> RGB
#plt.axis('off')
plt.imshow(rgb_img)
```

```
<matplotlib.image.AxesImage at 0x7fcaa45ed750>
```


```
spiral = cv2.imread(f"{watermarks_dir}/spiral.png")
rgb_spiral = spiral[:, :, ::-1]  # BGR ---> RGB
plt.imshow(rgb_spiral)
```

```
<matplotlib.image.AxesImage at 0x7fcaa4455210>
```



Wir testen unsere Funktion random_tile mit diesen beiden Bildern:

```
top_left, bottom_right = random_tile(image, spiral)
print(f"{top_left=}, {bottom_right=}")
print(f"{spiral.shape=}, {image.shape=}")
```

```
top_left=(577, 2569), bottom_right=(1367, 3363)
spiral.shape=(790, 794, 3), image.shape=(1800, 4000, 3)
```

Es ist zu erkennen, dass die Abmessungen der zufälligen Kachel dem Wasserzeichenbild entsprechen:

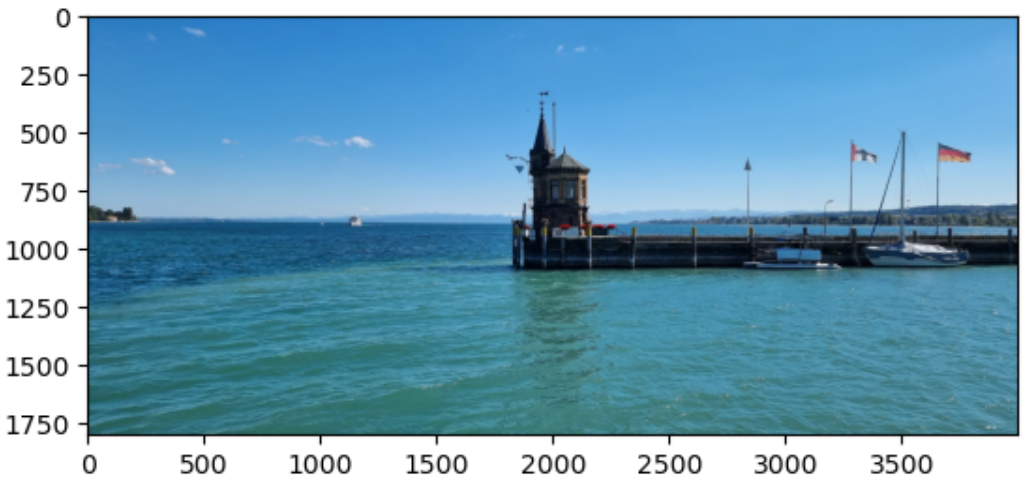
```
height_of_tile = bottom_right[0] - top_left[0]
width_of_tile = bottom_right[1] - top_left[1]
print( (height_of_tile, width_of_tile) == spiral.shape[:2])
```

```
True
```

Schauen wir uns ein weiteres Bild an. Diesmal geht es in den Hafen von Konstanz am Bodensee:

```
import matplotlib.pyplot as plt
import random
import numpy as np
import cv2

image2 = cv2.imread(f"{target_dir}/bodensee_area/konstanz.jpg")
rgb_img2 = image2[:, :, ::-1]  # BGR ---> RGB
plt.imshow(rgb_img2)
```

```
<matplotlib.image.AxesImage at 0x7fcaa44af150>
```



4.0.3 Wasserzeichen mit zusätzlichem Bild

Mit Hilfe der Funktion watermark_tile können wir ein Wasserzeichen auf den Bereich der Kachel setzen, der durch die Parameter oben_links, unten_rechts. Das Wasserzeichen soll

schwarz und weiß sein. Wenn das Wasserzeichen schwarz ist, werden die entsprechenden Pixel von imag2 genommen, wenn es weiß ist, werden die Pixel von imag1 genommen.

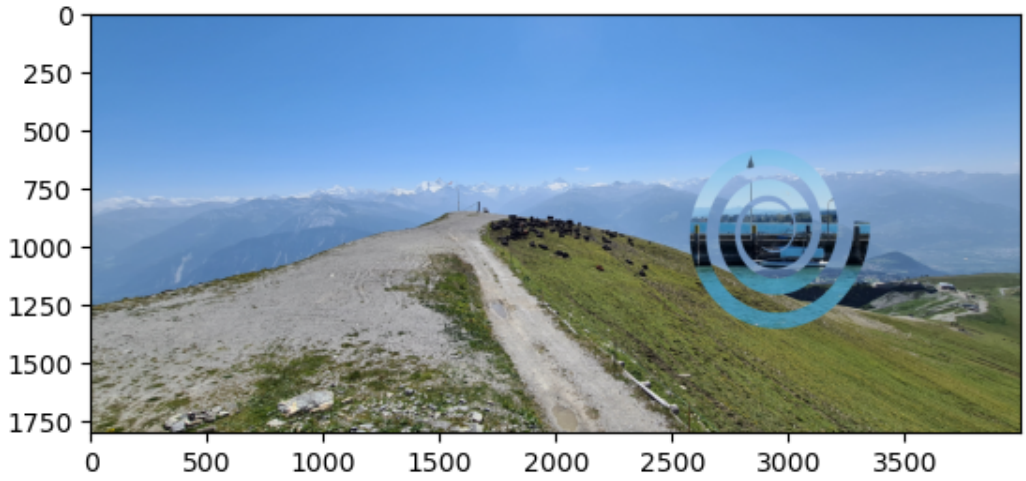
```
def watermark_tile(imag1, imag2, watermark_imag, top_left, bottom_right):
    """ The pixels of imag2 are used when the watermark_imag is black """
    r1, r2 = top_left[0], bottom_right[0]
    c1, c2 = top_left[1], bottom_right[1]
    tile = imag1[r1:r2, c1:c2]
    tile[:] = np.where(watermark_imag>(1, 1, 1), imag1[r1:r2, c1:c2],
    ↪  imag2[r1:r2, c1:c2])
```

Wir testen diese Funktion nun mit den Bildern ‘image’, ‘image2’ und ‘spiral’ als Wasserzeichenbild:

```
watermark_tile(image, image2, spiral,  top_left, bottom_right)
```

```
rgb_img = image[:, :, ::-1]  # BGR ---> RGB
plt.imshow(rgb_img)
```

```
<matplotlib.image.AxesImage at 0x7fcaa4340050>
```



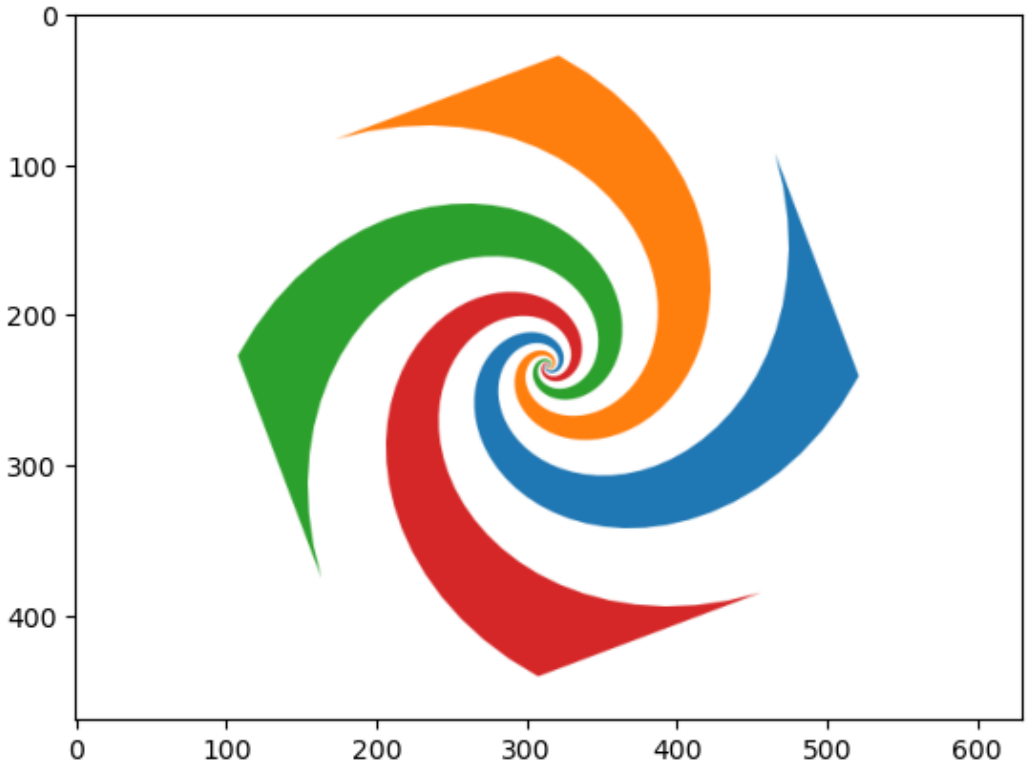
4.0.4 Wasserzeichen auf Kachel

Nun wollen wir ein Wasserzeichen direkt auf das Bild “imag1” setzen. Wir verwenden ein farbiges Bild als Wasserzeichen.

```
def object_on_tile(imag1, object_image, top_left, bottom_right):
    """ The object is put over the image """
    r1, r2 = top_left[0], bottom_right[0]
    c1, c2 = top_left[1], bottom_right[1]
    #print(f'{object_image=}')
    #tile = imag1[r1:r2, c1:c2]
    imag1[r1:r2, c1:c2] = np.where(object_image>(250, 250, 250),
                                   imag1[r1:r2, c1:c2],
                                   object_image)
```

```
watermark_image = cv2.imread(f"{watermarks_dir}/coloured_spiral.png")
rgb_watermark_image = watermark_image[:, :, ::-1]  # BGR ---> RGB
plt.imshow(rgb_watermark_image)
```

```
<matplotlib.image.AxesImage at 0x7fcaa4354ad0>
```




```
watermark_image.shape
```

```
(470, 630, 3)
```

```
image = cv2.imread(f"{target_dir}/bodensee_area/boat.jpg")
plt.imshow(image[:, :, ::-1])
```

```
<matplotlib.image.AxesImage at 0x7fcaa43b4ad0>
```


```
top_left, bottom_right = random_tile(image, watermark_image)
```

```
object_on_tile(image, watermark_image,  top_left, bottom_right)
```

```
rgb_img = image[:, :, ::-1]  # BGR ---> RGB
plt.imshow(rgb_img)
```

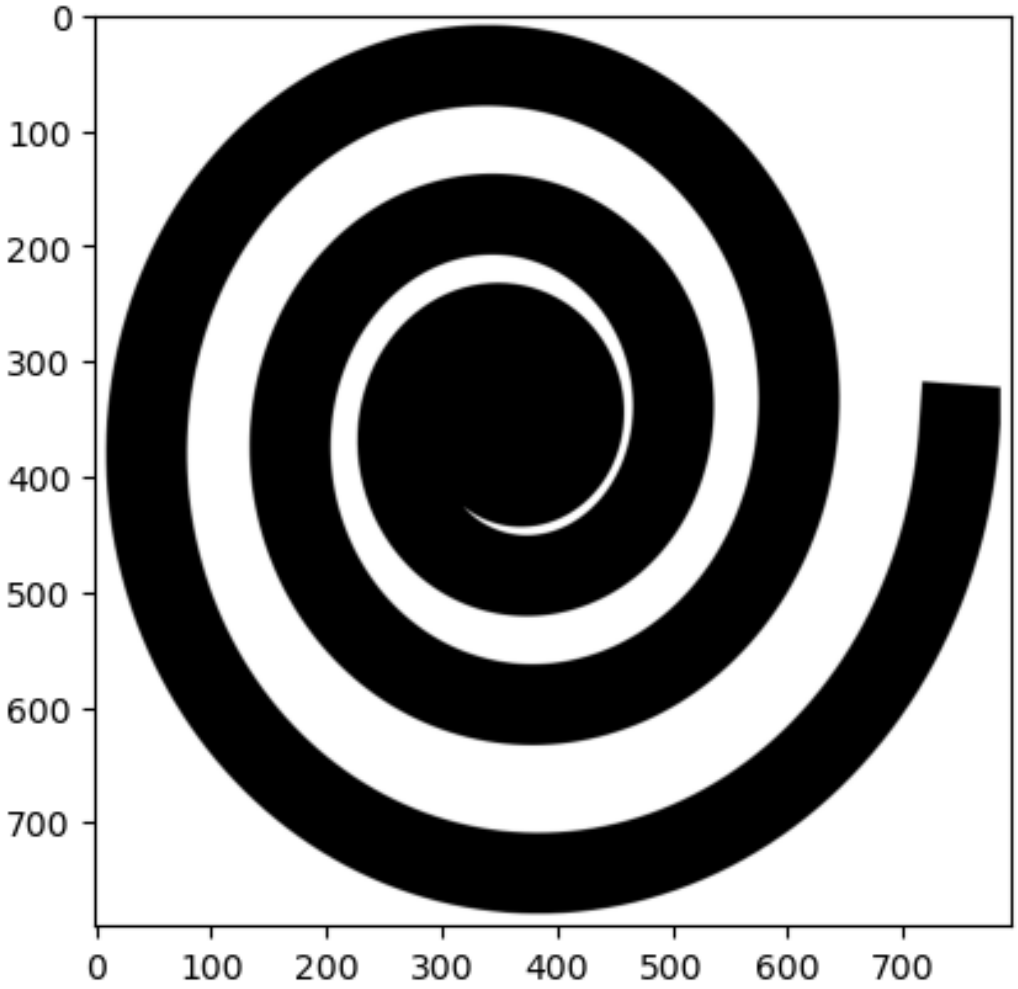
```
<matplotlib.image.AxesImage at 0x7fcaa42a7150>
```


```
image.shape
```

```
(1800, 4000, 3)
```

```
watermark_image = plt.imread(f"{watermarks_dir}/spiral.png")
watermark_image = watermark_image[:, :, :3]
#imag = cv2.imread("images/20210617_201910.jpg")
plt.imshow(watermark_image)
#cv2.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7fcaa4150bd0>
```



4.0.5 Kachel nahe beieinandeer

Wir schreiben nun eine Funktion new_close_tile, die zufällig eine neue Kachel in der Nähe einer bestehenden Kachel in der Umgebung `radius` findet.

```
import random

def new_close_tile(top_left, bottom_right,
                   img_width, img_height,
                   radius):
    """ finds randomly a new tile close to the given tile
    in a distance radius """
    row1, row2 = top_left[0], bottom_right[0]
    col1, col2 = top_left[1], bottom_right[1]
    x = random.randint(-radius-1, radius+1)
    y = random.randint(-radius-1, radius+1)
    tile_height = row2 - row1
    tile_width = col2 - col1
    row1_new, row2_new = row1 + y, row2 + y
    col1_new, col2_new = col1 + x, col2 + x
```

```
    while (row2_new > img_height) or row1_new < 0 or col1_new < 0 or col2_new >
    ↪  img_width:
        x = random.randint(-radius-1, radius+1)
        y = random.randint(-radius-1, radius+1)
        row1_new, row2_new = row1 + y, row2 + y
        col1_new, col2_new = col1 + x, col2 + x
    return ((row1_new, col1_new), (row2_new, col2_new))
```

Wir werden nun die Funktion `new_close_tile` testen:

```
# find a tile:
top_left, bottom_right = random_tile(image, watermark_image)
print(f"{top_left=} {bottom_right=}")
img_height, img_width, colours = image.shape
for i in range(5):
    top_left, bottom_right = new_close_tile(top_left, bottom_right,
                                            img_width, img_height,
                                            radius=4)
    print(f"New tile positions: {top_left=} {bottom_right=}")
```

```
top_left=(310, 710) bottom_right=(1100, 1504)
New tile positions: top_left=(310, 709) bottom_right=(1100, 1503)
New tile positions: top_left=(312, 712) bottom_right=(1102, 1506)
New tile positions: top_left=(313, 712) bottom_right=(1103, 1506)
New tile positions: top_left=(310, 716) bottom_right=(1100, 1510)
New tile positions: top_left=(315, 720) bottom_right=(1105, 1514)
```

4.0.6 Bilder mit beweglichen Wasserzeichen

```
n = 40
watermark_pics = f'{target_dir}/watermark_pics'
if os.path.exists(watermark_pics):
    shutil.rmtree(watermark_pics)
os.makedirs(watermark_pics)
watermark_image = cv2.imread(f"{watermarks_dir}/coloured_spiral.png")
watermark_image = watermark_image[:, :, :3]
image_orig = cv2.imread(f"{target_dir}/paths/pic_004.jpg")
height, width, colours = image_orig.shape
top_left, bottom_right = random_tile(image, watermark_image)
print(top_left, bottom_right)
for i in range(n, 0, -1):
    image = image_orig.copy()
    top_left, bottom_right = new_close_tile(top_left, bottom_right, width,
    ↪  height, 20)
    object_on_tile(image, watermark_image, top_left, bottom_right)
    cv2.imwrite(f"{watermark_pics}/pic_{i:03d}.png", image)
```

```
(318, 3152) (788, 3782)
```

```
def video_from_images(folder='.',
                      video_name = 'video.avi',
                      suffix='png',
                      prefix='pic_',
                      reverse=False,
```

```
                    length_of_video_in_seconds=None,
                    codec = cv2.VideoWriter_fourcc(*'DIVX')):
    """ The function creates a video from all the images with
    the suffix (default is 'png' and the prefix (default is 'pic_'
    in the folder 'folder'. If 'length_of_video_in_seconds' is set
    to None, it will be the number of images in seconds. If a positive
    value is given this will be the length of the video in seconds.
    The function assumes that the the shape of the first image is
    the one for all the images. If not a warning will be printed
    and the size will be adapted accordingly! """

    images = []
    for fname in glob.glob(f'{folder}/{prefix}*{suffix}'):
        images.append(fname)
    images.sort(reverse=reverse)
    if length_of_video_in_seconds is None:
        # each image will be shown for one second
        length_of_video_in_seconds = len(images)

    # calculate number of frames per seconds:
    frames_per_second = len(images) / length_of_video_in_seconds
    frame_size = get_frame_size(images[0])

    video = cv2.VideoWriter(video_name,
                            codec,
                            frames_per_second,
                            frame_size)

    for image in images:
        im = cv2.imread(image)
        height, width, layers = im.shape
        if (width, height) != frame_size:
            print(f'Warning: {image} resized from {(width,
            ↪  height)} to {frame_size}')
            im = cv2.resize(im, frame_size)
        video.write(im)
    cv2.destroyAllWindows()
    video.release()
```

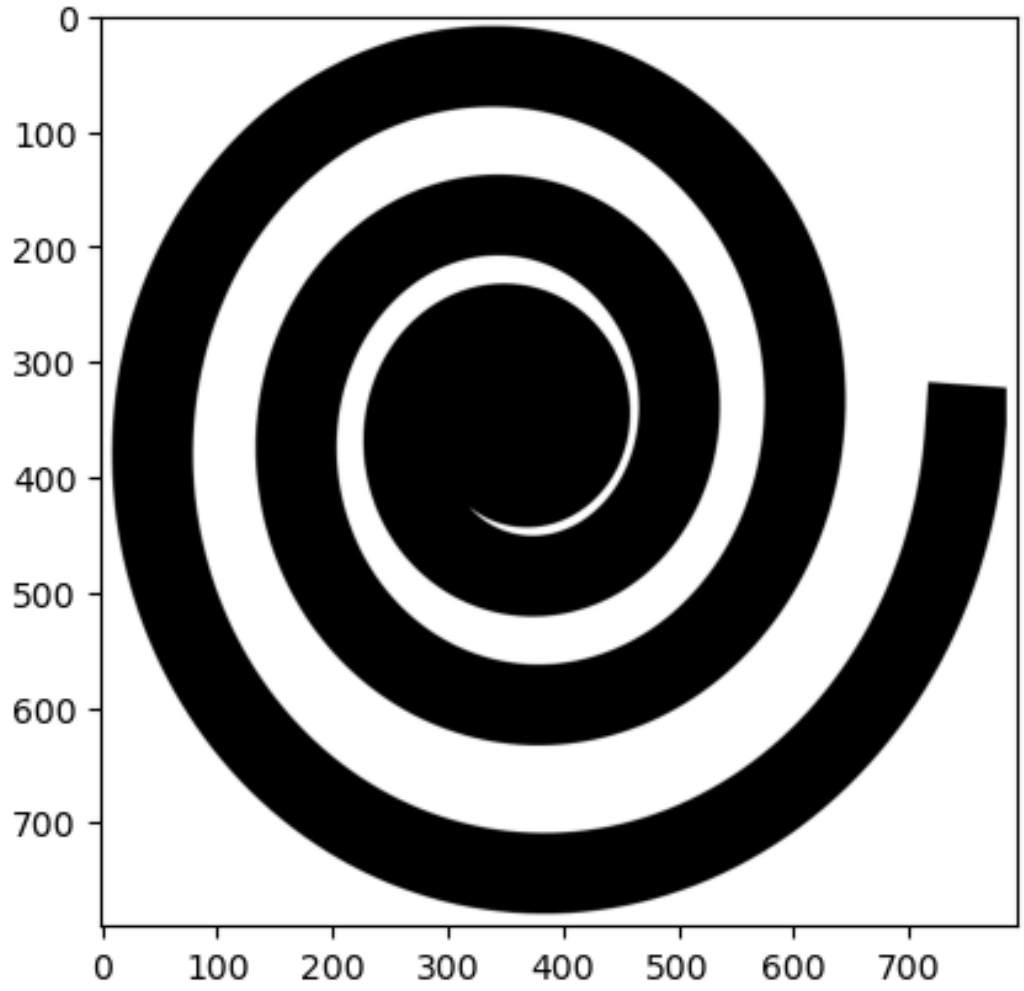
Wir haben 40 Bilder erzeugt und wir erzeugen nun ein Video mit einer Länge von 40 Sekunden:

```
import glob
video_from_images(watermark_pics,
                  f'{target_dir}/colour_spiral.avi',
                  'png',
                  length_of_video_in_seconds = 40)
```

Für das nächste Video benutzen wir eine schwarz-weiße Spirale. Wir laden sie zuerst:

```
watermark_image = cv2.imread(f"{watermarks_dir}/spiral.png")
plt.imshow(watermark_image[:,:,::-1])
```

```
<matplotlib.image.AxesImage at 0x7fcaa41a8dd0>
```




```
n = 40
watermark_pics = f'{target_dir}/watermark_pics2'
if os.path.exists(watermark_pics):
    shutil.rmtree(watermark_pics)
os.makedirs(watermark_pics)

image_orig = cv2.imread(f"{target_dir}/paths/pic_002.jpg")
imag_tinted = image_orig // 2
for i in range(n, 0, -1):
    image = image_orig.copy()
    top_left, bottom_right = random_tile(image, watermark_image)
    watermark_tile(image, imag_tinted, watermark_image,  top_left,
    ↪  bottom_right)
    cv2.imwrite(f"{watermark_pics}/pic_{i:03d}.png", image)
```

```
import glob
video_from_images(watermark_pics,
                  f'{target_dir}/spiral.avi',
                  'png',
                  length_of_video_in_seconds=40)
```

Die Videos, die wir in diesem Tutorial erstellt haben, kann man sich hier anhören:

- spiral.avi
- colour_spiral.avi

Ich habe diese Technik verwendet, um Videos für Musikkompositionen von mir zu erstellen. Die Videos sind bei YouTube zu finden:

- Bernd Klein - Winding Paths
- Bernd Klein - Days Like This

Musikstück für Klavier, Saxophon, Oboe und Schlagzeug Die Tonspur habe ich mit dem Linux-Programm kdenlive hinzugefügt.

Alle meine Videos, einschließlich eines Python-Videos, findet man unter Bernd Klein's Videos

Weitere Bücher von Bernd Klein

Die folgenden Verlagstitel sind eigenständige Veröffentlichungen beim Carl Hanser Verlag. Diese automatisch erzeugte PDF-Ausgabe von `python-kurs.eu` ist keine Hanser-Veröffentlichung.

Einführung in Python 3

Für Ein- und Umsteiger

Bernd Klein

ISBN: 978-3-446-46379-0

4. Auflage, 2021

Mehr beim Carl Hanser Verlag

Numerisches Python

Arbeiten mit NumPy, Matplotlib und Pandas

Bernd Klein

ISBN: 978-3-446-48584-6

3. Auflage, 2025

Mehr beim Carl Hanser Verlag

Funktionale Programmierung mit Python

Funktionale Konzepte praxisnah in Python einsetzen

Bernd Klein, Philip Klein

ISBN: 978-3-446-48191-6

1. Auflage, 2025

Mehr beim Carl Hanser Verlag

Hinweis zur Ausgabe

Diese Ausgabe wurde automatisiert aus den Quellen von `python-kurs.eu` erzeugt. Die Online-Fassung kann aktueller sein als diese PDF-Ausgabe.