exercise.01

April 30, 2020

1 Aufgabenblatt 1

Patrick Zierahn (7065799) Thorben Janssen (7047766)

1.1 Aufgabe 1 — NumPy und Matplotlib Tutorial

1. Wie kann die Anzahl der Zeilen und Spalten eines NumPy-Arrays bestimmt werden?

An NumPy-Arrays lässt sich .shape aufrufen. Dies ist ein Tuple das die Dimension zurück liefert.

2. Was unterscheidet die Funktionen numpy.array und numpy.zeros?

Mit numpy.array läst sich eine Matrix erstellen. Mit numpy.zeros((n, m)) läst sich eine n*m Matrixs erstellen die mit nullen gefühlt ist.

3. Wie kann der Datentyp eines gegebenen NumPy-Arrays ermittelt werden?

Die Funktion type(a) gibt Auskunft darüber um was für einen Datentyp es sich handelt.

4. Was bedeutet .T hinter dem Variablennamen eines NumPy-Arrays? Beispielsweise: a.T

Das T bedeutet das die Matrix transponiert wird.

5. Was machen die Funktionen xlabel und ylabel aus matplotlib.pyplot?

Mit matplotlib lassen sich Grafiken und Charts generieren. xlabel und ylabel setzt die x- und y-Axsen beschriftung.

1.2 Aufgabe 2 — Erste Schritte in NumPy

[1]:
```
import numpy as np

# 1. Erzeugt zunächst einen Vektor u als 1D-Array mit 100 Nullen
u = np.zeros((100))
print("u =", u)

# 2. Erzeugt zudem einen Vektor v als 1D-Array aus der Liste:␣
↪[0,1,2,3,4,5,6,7,8,9,10,11].
```

```
v = np.array([0,1,2,3,4,5,6,7,8,9,10,11])
print("v =", v)

# 3. Formt den Vektor v nun in eine Matrix m (2D-Array) um, die aus drei Zeilen␣
 ↪und vier Spalten besteht.
# Tipp: NumPy kennt eine Funktion, um die Form (engl. shape) eines Arrays zu␣
 ↪ändern.
m = v.reshape((3,4))
print("m =", m)

# 4. Multipliziert alle Einträge der Matrix m mit dem konstanten Faktor 1.2.
m = m * 1.2
print("m =", m)

# 5. Ändert nun den Datentyp der Matrix m auf np.int.
# Tipp: Nutzt die Methode astype eines NumPy-Arrays.
m = m.astype(np.int)
print("m =", m)

# 6. Multipliziert alle Einträge der Matrix m erneut mit dem konstanten Faktor␣
 ↪1.2.
# Von welchem Datentyp ist jetzt das Ergebnis?
m = m * 1.2
print("m =", m)
print("type(m) =", type(m))

# 7. Berechnet das Hadamard-Produkt (elementweise Multiplikation)
# der Matrix m mit sich selbst (m m).
m = m * m
print("m =", m)
```

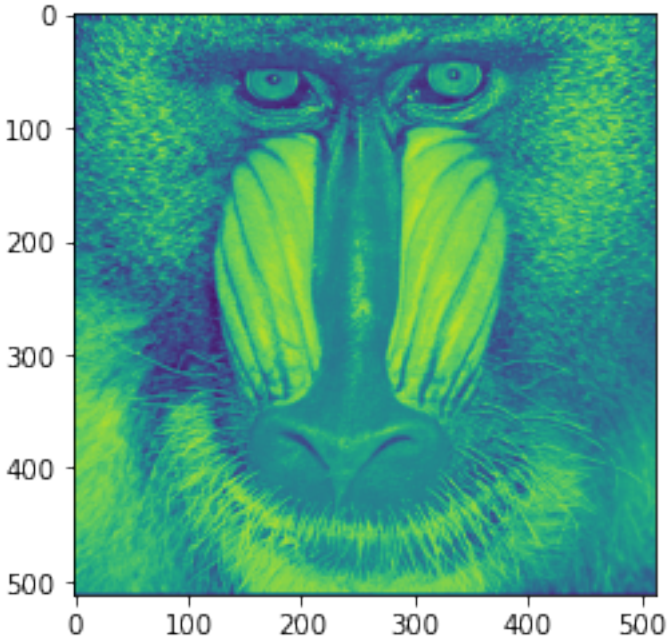
```
u = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0.]
v = [ 0  1  2  3  4  5  6  7  8  9 10 11]
m = [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
m = [[ 0.   1.2  2.4  3.6]
 [ 4.8  6.   7.2  8.4]
 [ 9.6 10.8 12.  13.2]]
m = [[ 0  1  2  3]
 [ 4  6  7  8]
 [ 9 10 12 13]]
m = [[ 0.   1.2  2.4  3.6]
```

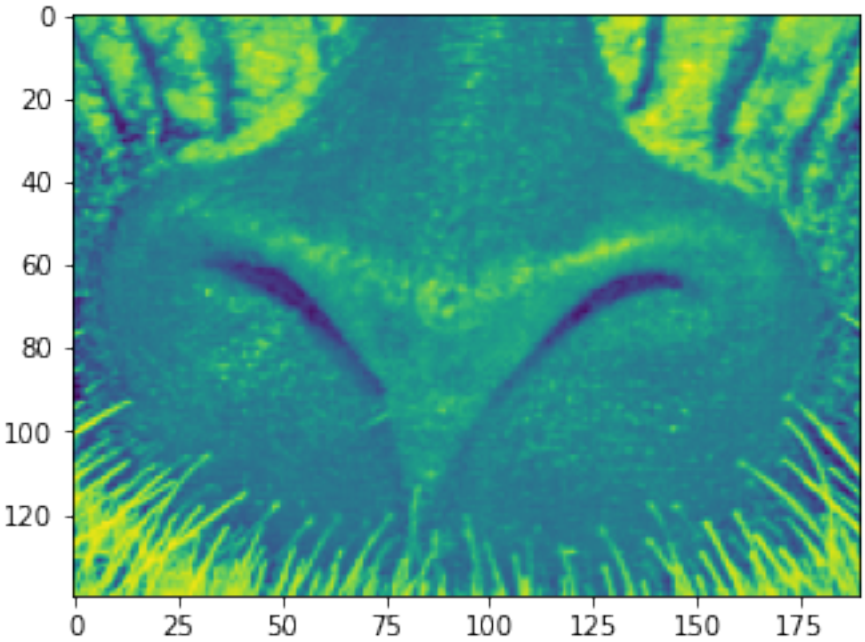
```
 [ 4.8  7.2  8.4  9.6]
 [10.8 12.  14.4 15.6]]
type(m) = <class 'numpy.ndarray'>
m = [[  0.     1.44   5.76  12.96]
 [ 23.04  51.84  70.56  92.16]
 [116.64 144.   207.36 243.36]]
```

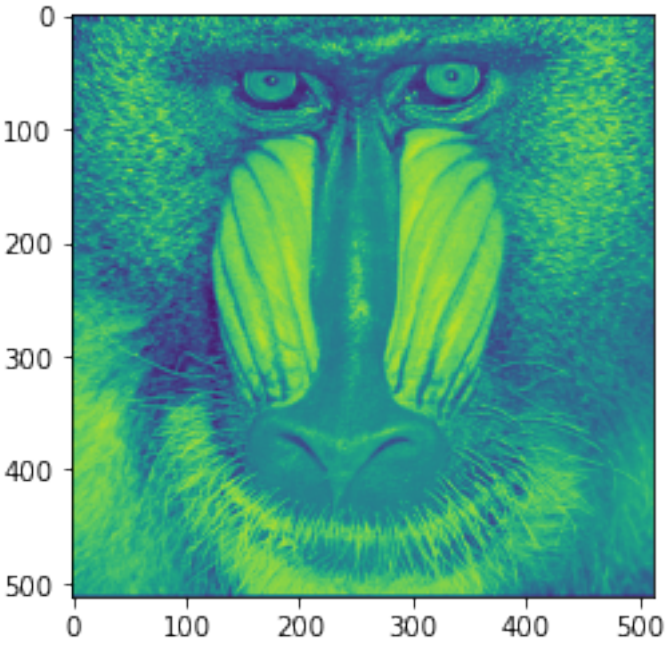
1.3 Aufgabe 3 — Basisoperationen mit Bildern

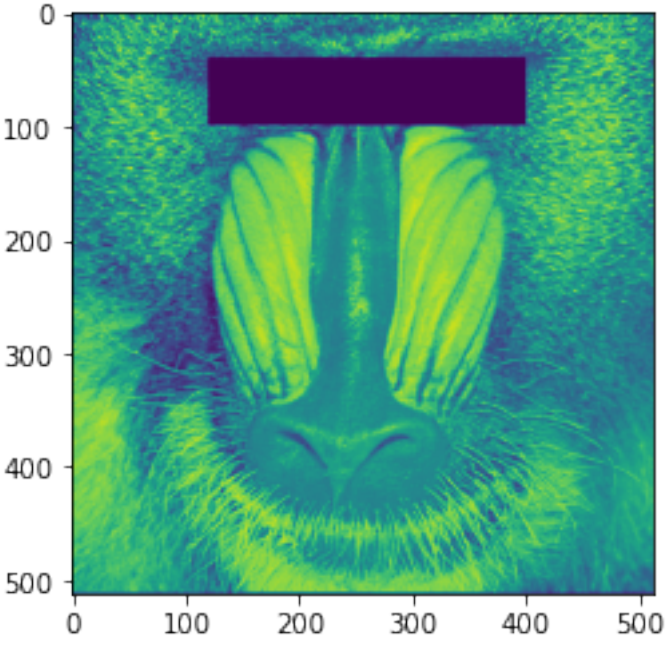
[2]:
```python
import skimage.io
import matplotlib.pyplot as plt

# 1. Speichert euch das Bild mandrill.png aus dem Moodle ab
# und ladet es in Pythonmithilfe der Funktion skimage.io.imread.
mandrill = skimage.io.imread("mandrill.png")

# 2. Zeigt das Bild mithilfe von Matplotlib an.
plt.figure()
plt.imshow(mandrill)

# 3. Erstellt ein neues Array als Bildausschnitt, das, wie in
# Abbildung 1a visualisiert, etwa die Nasen- spitze des Mandrill beinhaltet.
nose = mandrill[310:450, 150:340]
plt.figure()
plt.imshow(nose)

# 4. Speichert den Bildausschnitt mit der Funktion skimage.io.imsave ab.
skimage.io.imsave("exercise.01.03.04.png", nose)

# 5. Erstellt eine Kopie des Bildes und setzt ein einziges
# Pixel (Arrayelement bzw. Bildelement) auf schwarz.
# Lasst euch das manipulierte Bild anzeigen.
# Erkennt man den Unterschied? (kein Antworttext nötig)
copy_mandrill = mandrill.copy()
copy_mandrill[0] = 0
plt.figure()
plt.imshow(copy_mandrill)

# 6. Erstellt eine weitere Kopie des Bildes und setzt den
# gesamten Bereich der Augendes Mandrills auf schwarz.
# Lasst euch das manipulierte Bild erneut anzeigen.
# Erkennt man nun den Unterschied? (kein Antworttext nötig)
copy_mandrill2 = mandrill.copy()
copy_mandrill2[40:100, 120:400] = 0
plt.figure()
plt.imshow(copy_mandrill2)
```

[2]: <matplotlib.image.AxesImage at 0x7efd6e05b190>






0
100
200
300
400
500
0
100
200
300
400
500

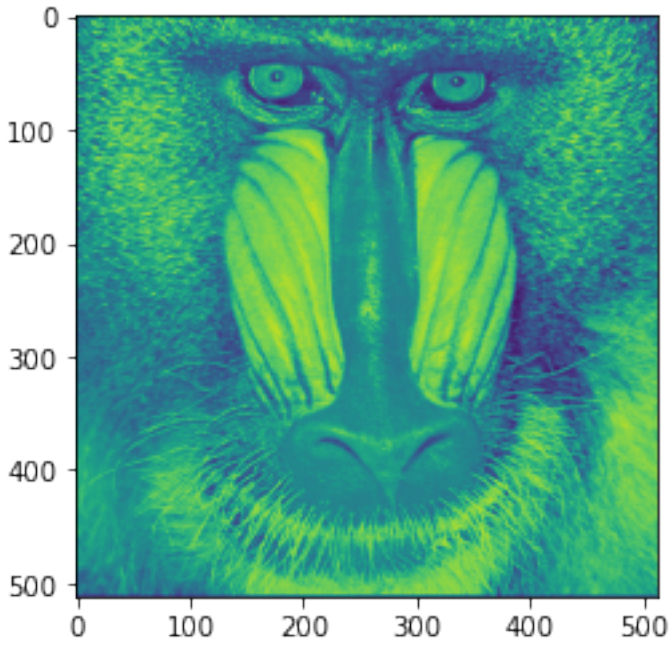

0
100
200
300
400
500
0
100
200
300
400
500

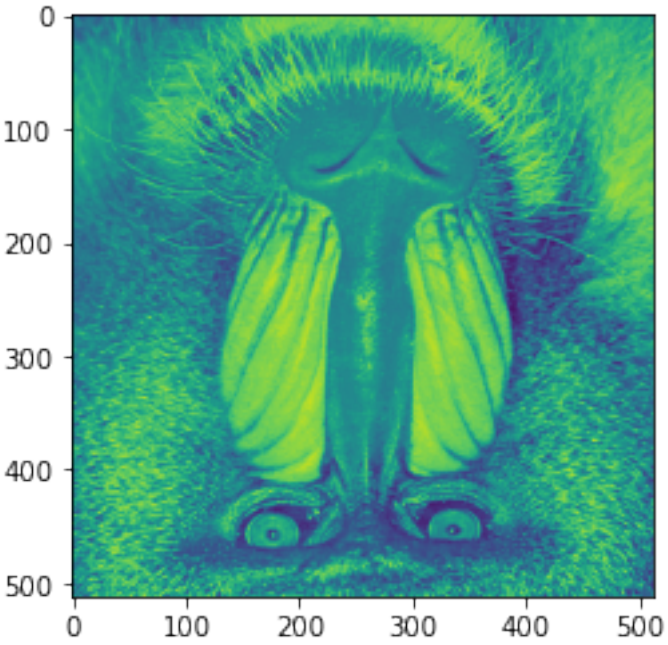
1.4 Aufgabe 4 — Fortgeschrittene Bearbeitung von Bildern

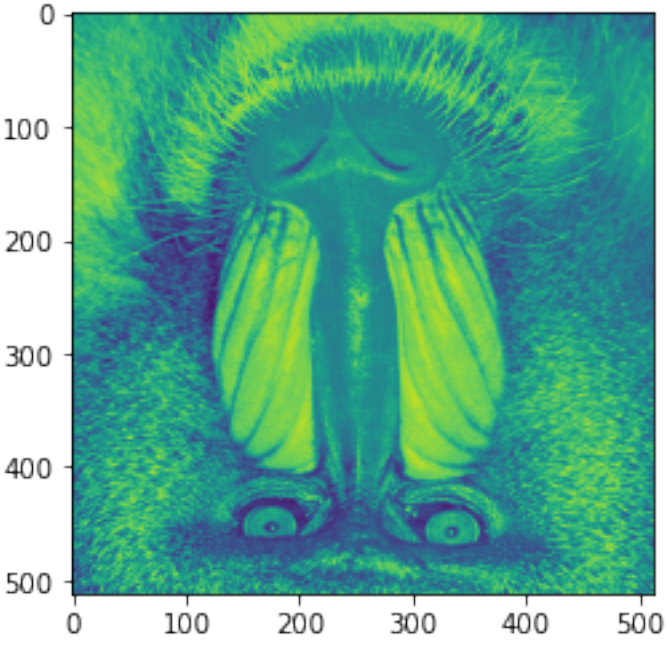
1.4.1 1. Erstellt ein neues Bild basierend auf drei Variationen des Bildes mandrill.png aus dem Moodle sowie dem Originalbild.

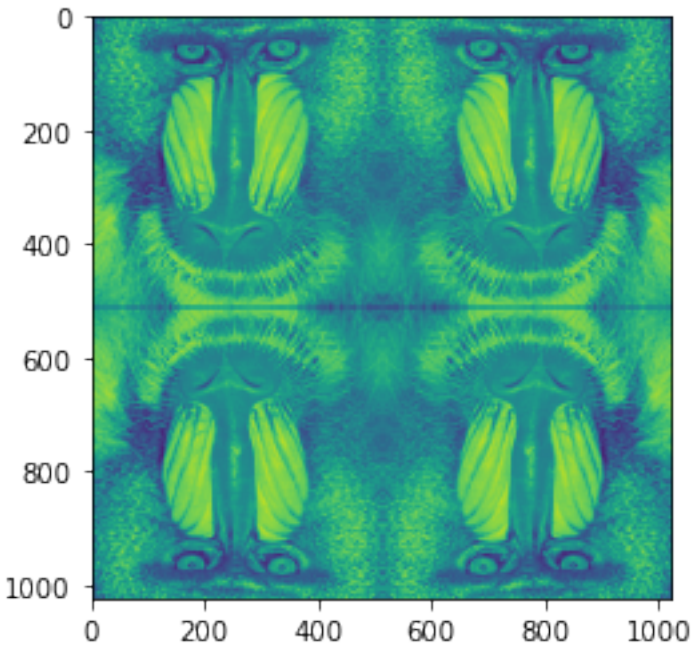
[3]:
```
import numpy
import skimage.io
import matplotlib.pyplot as plt

# a) Ladet dazu zunächst erneut das Bild mandrill.png aus dem Moodle in Python.
mandrill = skimage.io.imread("mandrill.png")

# b) Spiegelt das Bild an der vertikalen Achse (linke und rechte Seite tasuchen)
# und speichert das Ergebnis in eine neue Variable als erste Variation.
mandrill_fliplr = numpy.fliplr(mandrill)
plt.figure()
plt.imshow(mandrill_fliplr)
skimage.io.imsave("exercise.01.04.01.b.png", mandrill_fliplr)

# c) Spiegelt das Originalbild (mandrill.png) nun an der horizontalen
# Achse und speichert das Ergebnis erneut in einer neuen Variable.
# Dies ist die ist die zweite Variation.
mandrill_flip = numpy.flip(mandrill)
plt.figure()
plt.imshow(mandrill_flip)
skimage.io.imsave("exercise.01.04.01.c.png", mandrill_flip)

# d) Führt jetzt beide Spiegelungen nacheinander auf
# dem Originalbild aus speichert das Ergebnis wiederum
# in einem neuen Bild als dritte Variation.
mandrill_fliplr_flip = numpy.fliplr(numpy.flip(mandrill))
plt.figure()
plt.imshow(mandrill_fliplr_flip)
skimage.io.imsave("exercise.01.04.01.d.png", mandrill_fliplr_flip)

# e) Erzeugt nun ein neues Bild, das doppelt so hoch und breit ist
# wie das Bild mandrill.png. Setzt in jeden Quadranten des neuen
# Bildes eine der vier Versionen des Bildes ein.
# Oben links soll das Origianlbild zu sehen sein,
# rechts daneben das an der vertikalen Achse gespiegelte Bild,
# links unten das an der horizontalen Achse gespiegelte Bild
# und rechts unten das an beiden Achsen gespiegelte Bild.
# Das Ergebnis sollte etwa wie in Abbildung 1b aussehen.
# Tipp: Zum Erzeugen von Arrays bzw. Bildern gibt es in NumPy mehrere␣
 ↪Funktionen neben np.array.

width = mandrill.shape[0]
```

```
height = mandrill.shape[1]
composed_image = numpy.zeros((width * 2, height * 2))

# left top
composed_image[:height, :width] = mandrill

# top right
composed_image[:height, width:] = mandrill_fliplr

# left bottom
composed_image[height:, :width] = mandrill_fliplr_flip

# bottom right
composed_image[height:, width:] = mandrill_flip

# Remove pesky error message
composed_image = composed_image.astype(numpy.uint8)

plt.figure()
plt.imshow(composed_image)
skimage.io.imsave("exercise.01.04.01.e.png", composed_image)
```




0
100
200
300
400
500
0
100
200
300
400
500

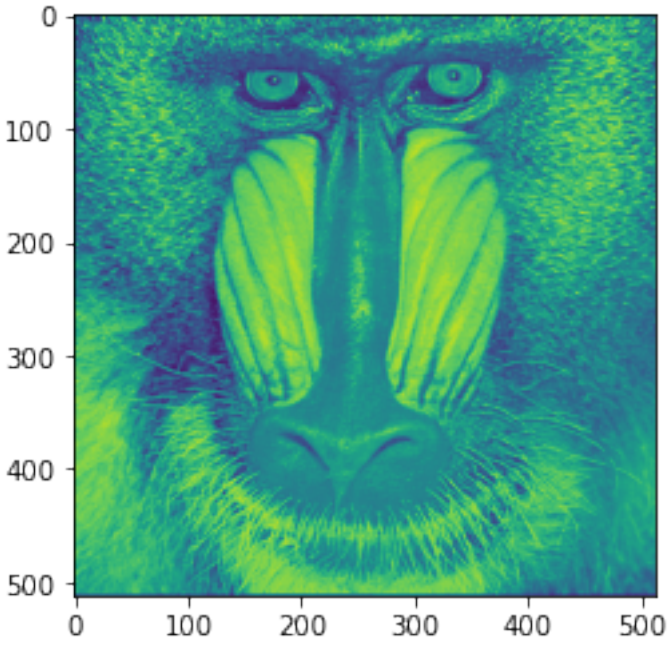

0
100
200
300
400
500
0
100
200
300
400
500

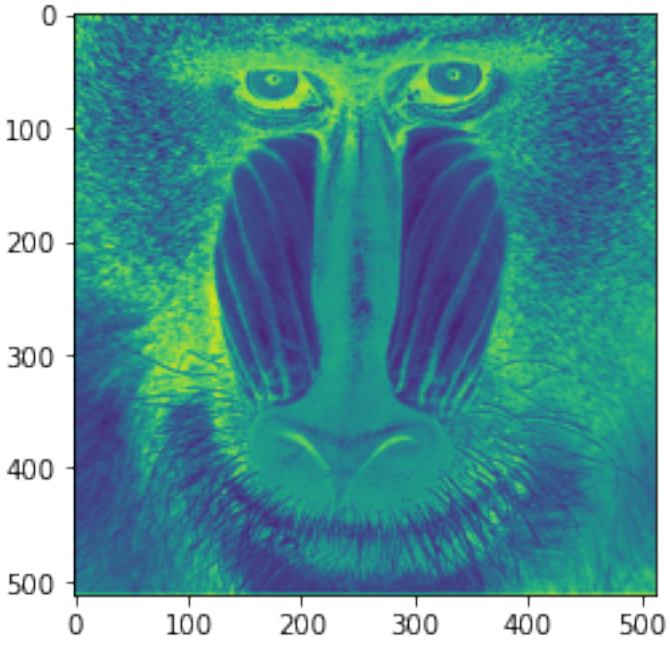


1.4.2 2. ErzeugtnuneinNegativdesOriginalbilds(mandrill.png).IneinemNegativsindalleHelligkeitswerte umgedreht. D.h. schwarz wird zu weiß, weiß wird zu schwarz und alle Werte dazwischen werden ebenso gedreht. Lasst euch das Bild zur Kontrolle anzeigen.

[4]:
```
import numpy
import skimage.io
import matplotlib.pyplot as plt

mandrill = skimage.io.imread("mandrill.png")
plt.figure()
plt.imshow(mandrill)

inverse_matrix = numpy.ones(mandrill.shape) * 0xff

mandrill_mattrix = numpy.array(mandrill)
mandrill_mattrix = mandrill_mattrix - inverse_matrix
mandrill_mattrix = numpy.absolute(mandrill_mattrix)

# Remove pesky error message
mandrill_mattrix = mandrill_mattrix.astype(numpy.uint8)

plt.figure()
plt.imshow(mandrill_mattrix)
```

```
skimage.io.imsave("exercise.01.04.02.png", mandrill_mattrix)
```

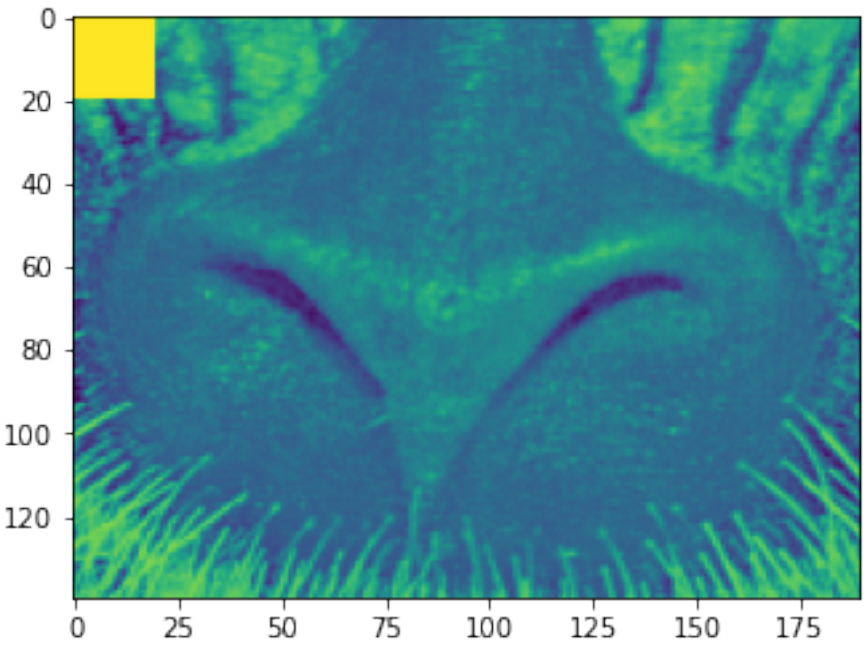


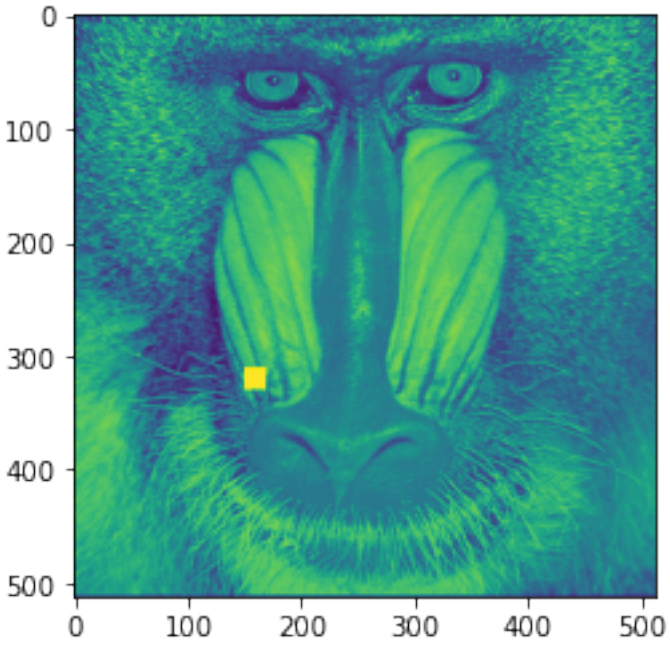


1.4.3 3. Schneidet erneut die Nasenspitze des Mandrills wie in Abbildung 1a dargestellt als neues Bild aus. Ändert nun in dem neu erzeugten Ausschnitt ein Pixel und findet heraus, ob diese Änderung auch im Originalbild durchgeführt wird.

[5]:
```
import skimage.io
import matplotlib.pyplot as plt

mandrill = skimage.io.imread("mandrill.png")
nose = mandrill[310:450, 150:340]

nose[:20, :20] = 0xff
plt.figure()
plt.imshow(nose)

plt.figure()
plt.imshow(mandrill)

# Antwort: die veränderungen sind auch im Originalbild sichtbar!
```

[5]:
```
<matplotlib.image.AxesImage at 0x7efd6dcee590>
```



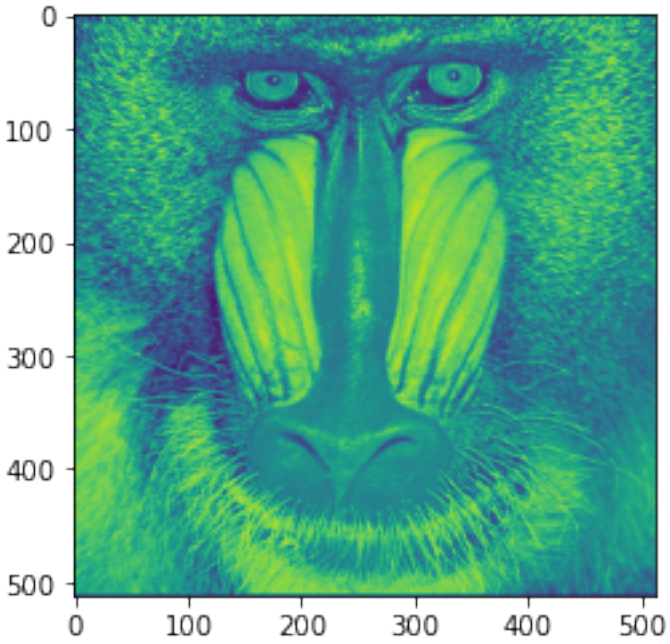


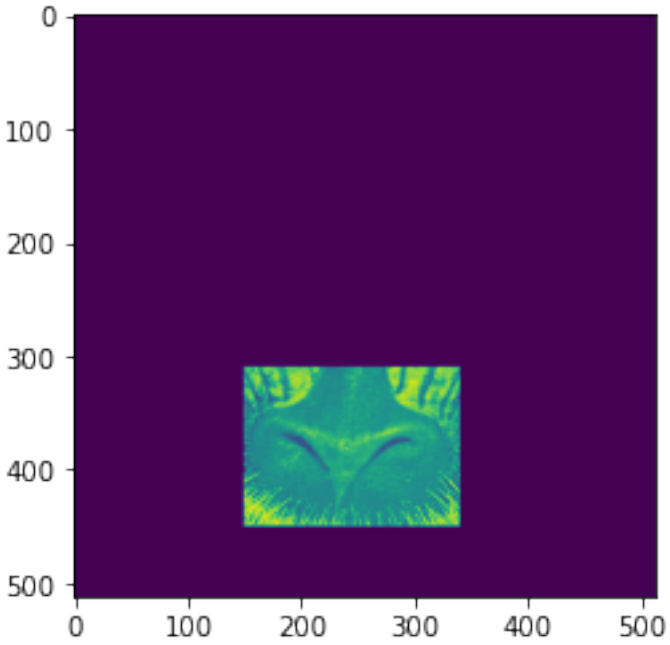
1.4.4 4. Erstellt euch nun eine Maske für den Bereich der Nasenspitze des Mandrills.

[6]:
```
import numpy
import skimage.io
import matplotlib.pyplot as plt

mandrill = skimage.io.imread("mandrill.png")
plt.figure()
plt.imshow(mandrill)

nose = mandrill[310:450, 150:340]

# a) Erzeugt dazu zunächst ein neues Bild, das nur aus Nullen besteht,
# und die gleiche Größe wie das Originalbild hat. Dieses Bild wird eure Maske.
mask = numpy.zeros(mandrill.shape)

# b) Setzt nun in der Maske alle Pixel, die in dem Bereich liegen, der die␣
 ↪Nasenspitze beinhaltet, auf 1.
mask[310:450, 150:340] = numpy.ones(nose.shape)

# c) Verrechnet anschließend Maske und Originalbild,
# sodass der Bereich außerhalb der Nasenspitze schwarz
# ist und nur im Bereich der Nasenspitze das eigentliche
# Bild zu sehen ist. Das Ergebnis sollte etwa wie in Abbildung 1c aussehen.
```

```
mandrill = mandrill * mask
plt.figure()
plt.imshow(mandrill)
```

[6]: <matplotlib.image.AxesImage at 0x7efd6dc5db10>





1.5 Aufgabe 5 — Broadcasting vs. Schleifen in NumPy

[7]:
```python
import time
import numpy
import skimage.io

# 1. Schreibt eine Python-Funktion, die ein Bild als
# NumPy-Array annimmt und für jeden Arrayeinträge bzw.
# Pixel prüft, ob der Pixelwert zwischen 99 und 200 liegt.
# Umgesetzt werden soll dies durch einer Iteration über
# das Bild mittels zweier for-Schleifen.
def check_pixel_for(picture: numpy.array) -> bool:
    for iny in range(picture.shape[0]):
        for inx in range(picture.shape[1]):
            pixel = mandrill_matrix[iny][inx]

            if pixel < 99 or pixel > 200:
                return False

    return True

# 2. Schreibt eine zweite Funktion für selbigen Zweck.
# Diesmal jedoch mit Broadcasting, d.h.komplett
# ohne die Benutzung von Schleifen.
def check_pixel_broadcasting(picture: numpy.array) -> bool:
    return 99 < picture.all() < 200

# 3. Messt wie lange 100 Aufrufe der beiden Funktionen
# jeweils auf das Bild mandrill.png aus dem Moodle dauern.
# Um die beiden Funktionen aufzurufen, dürft ihr natürlich
# Schleifen verwenden. Wie groß ist der Unterschied?

mandrill = skimage.io.imread("mandrill.png")
mandrill_matrix = numpy.array(mandrill)

tic1 = time.time()
for _ in range(100):
    check_pixel_for(mandrill_matrix)

toc1 = time.time()
print("for time =", toc1 - tic1)

tic2 = time.time()
```

```
for _ in range(100):
    check_pixel_broadcasting(mandrill_matrix)

toc2 = time.time()
print("Broadcasting time =", toc2 - tic2)

# Antwort: Signifikanter unterschied zwischen beiden Versionen.
```

```
for time = 0.0012307167053222656
Broadcasting time = 0.005385875701904297
```

1.6 Aufgabe 6 — Bildstatistiken in NumPy

[8]:
```
import numpy
import skimage.io

# 1. Ladet erneut das Bild mandrill.png in Python.
mandrill = skimage.io.imread("mandrill.png")
mandrill_matrix = numpy.array(mandrill)

# 2. Ermittelt Minimum, Maximum, Durchschnitt und
# die Standardabweichung des Bildes mit hilfe
# der entsprechenden NumPy-Funktionen.
print("Minimum", mandrill_matrix.min())
print("Maximum", mandrill_matrix.max())
print("Durchschnitt", numpy.average(mandrill_matrix))
print("Standardabweichung", numpy.std(mandrill_matrix))

# 3. Lokalisiert das Minimum und das Maximum im Bild.
# Gebt dazu die x- und y-Koordinate eines Minimums und
# Maximums auf der Konsole aus.
# Tipp: Schaut in die Dokumentation der entsprechenden
# Funktionen, um zu erfahren, wie ihr aus dem einen Ergebnisindex
# zwei Koordinaten generiert.
print("Minimum X Y =", numpy.unravel_index(numpy.argmin(mandrill_matrix),␣
 ↪mandrill_matrix.shape))
print("Maximum X Y =", numpy.unravel_index(numpy.argmax(mandrill_matrix),␣
 ↪mandrill_matrix.shape))

# 4. Zählt wie viele Pixel es mit einem
# geraden bzw. einem ungeraden Wert im Bild gibt.
# Nutzt auch hierfür wiederum keine Schleifen!

even_pixels = numpy.mod(mandrill_matrix, 2)==0
print("all_pixels_count", mandrill_matrix.size)
print("even_pixels_count", len(mandrill_matrix[even_pixels]))
```

```
print("odd_pixels_count", mandrill_matrix.size -␣
 ↪len(mandrill_matrix[even_pixels]))

# 5. Ermittelt außerdem alle Koordinaten andenen Pixel mit
# geradem Wert lokalisiert sind. Die Form der Koordinaten bspw.
# ((x1,y1),(x2,y2),(x3,y3),...) oder ((x1,x2,x3,...),(y1,y2,y3,...)) ist␣
 ↪unerheblich.
print("even_pixels", mandrill_matrix[even_pixels])
```

```
Minimum 0
Maximum 232
Durchschnitt 129.53657913208008
Standardabweichung 43.27355497483949
Minimum X Y = (511, 177)
Maximum X Y = (74, 295)
all_pixels_count 262144
even_pixels_count 131133
odd_pixels_count 131011
even_pixels [56 74 92 …  4  4  4]
```