Bildsegmentierung

Bei der Bildsegmentierung soll das Bild in sinnvolle Bildteile (Segmente) unterteilt werden. Diese Segmente bilden anschließend die Grundlage für die weitere Verarbeitung.

In [1]:

```
import pydicom
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv2 # opencv-python installieren
import os
import cv2 # opencv-python installieren
from scipy.ndimage import center_of_mass
from skimage.measure import label, regionprops
from skimage.segmentation import clear_border
import plotly.express as px
import ipywidgets as wg
from IPython.display import display
from PIL import Image
```

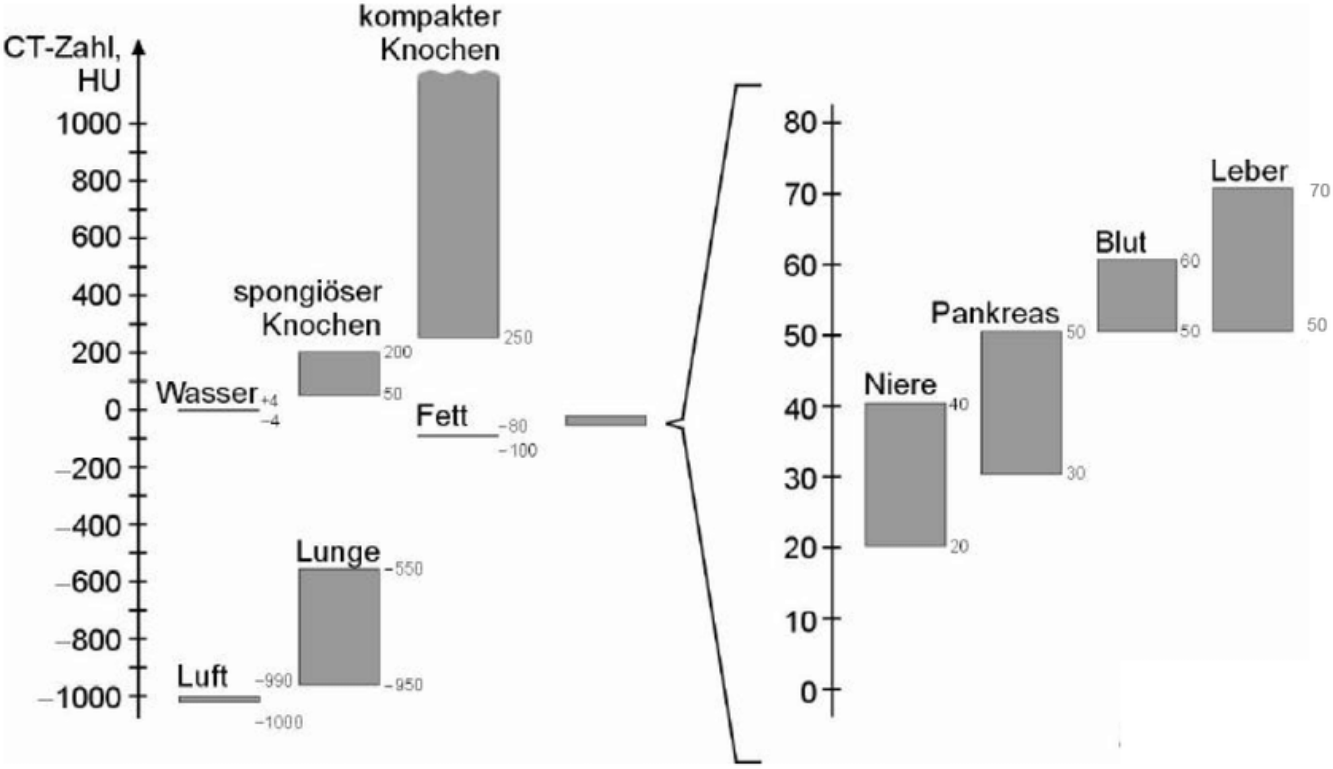
Hounsfield-Einheiten

Im folgenden Beispiel soll es darum gehen, die Lunge aus einer CT-Aufnahme zu segmentieren. Um das weitere Vorgehen zu verstehen, ist es hilfreich zu wissen, was es mit Hounsfield-Einheiten und der Hounsfield-Skala auf sich hat. Die Hounsfield-Skala gibt Auskunft über die Abschwächung der Röntgenstrahlung bei einer Computertomographie. Als Referenzwerte nutzt man die Relation zu Luft und Wasser.

DIe sogenannte CT-Zahl wird anhand des linearen Schwächungskoeffizienten des betrachteten Gewebes im Verhältnis zu Wasser und Luft berechnet.

$$[\text{CT-Zahl}]\,(\mu_{\text{Gewebe}}) := \frac{\mu_{\text{Gewebe}} - \mu_{\text{Wasser}}}{\mu_{\text{Wasser}} - \mu_{\text{Luft}}} \cdot 1000\,\text{HU}$$

Hier noch eine Hounsfield-Skala für verschiedene Gewebearten:



Laden der Daten und Definition der Hilfsfunktionen

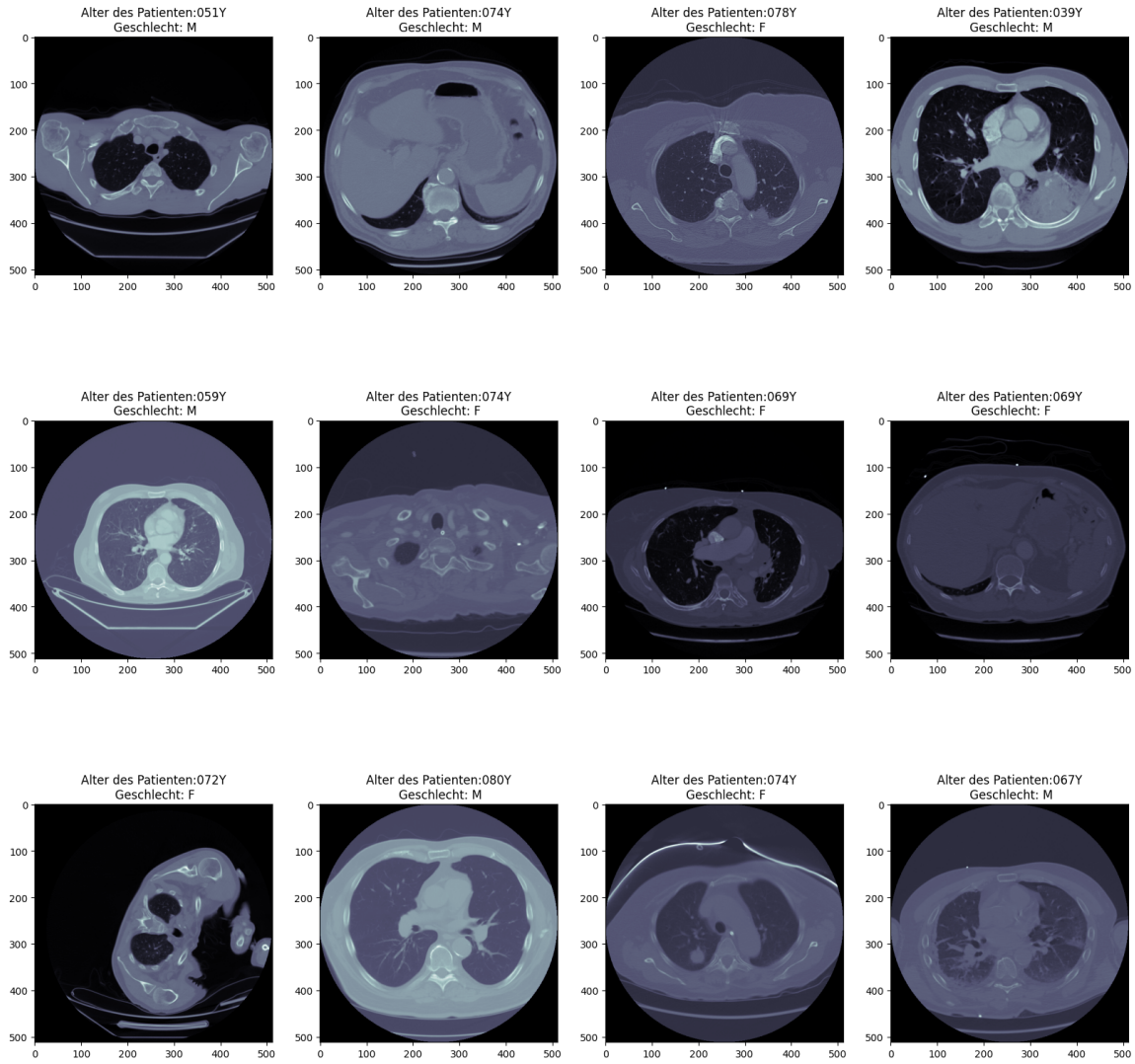
In [2]:

```
def load_scan(path):
    slices = [pydicom.read_file(path+"/"+s) for s in os.listdir(path) ]
    slices.sort(key = lambda x: int(x.AcquisitionNumber))
    try:
        slice_thickness = np.abs(slices[0].ImagePositionPatient[2] - slices[1].ImagePosit
    except:
        slice_thickness = np.abs(slices[0].SliceLocation - slices[1].SliceLocation)

    for s in slices:
        s.SliceThickness = slice_thickness
        s.SamplesPerPixel = 1

    return slices

def get_pixels_hu(scans):
    image = np.stack([s.pixel_array for s in scans[:100]])
    # Konvertierung zu int16
    image = image.astype(np.int16)
    # Pixel außerhalb des Scans auf Wert 1 setzen
    # Intercept normalerweise -1024, also liegt Luft ca. bei 0
        # (Interceptor: Wert im Verhältnis zwischen gespeicherten Werten in Pixeldaten un
        #                                                                  spezifizi
    image[image == -2000] = 0
    # Konvertierung der Pixelwerte zu Hounsfield-Einheiten (Hounsfield units - HU)
    intercept = scans[0].RescaleIntercept
    slope = scans[0].RescaleSlope

    if slope != 1:
        image = slope * image.astype(np.float64)
        image = image.astype(np.int16)

    image += np.int16(intercept)

    return np.array(image, dtype=np.int16)

path =  "dicom_dir"
patient = load_scan(path)
imgs = get_pixels_hu(patient)
```

Anzeigen einiger Beispielbilder

In [3]:

```
fig = plt.figure(figsize=(20,20))
for num,image in enumerate(imgs[:12]):
    ax = fig.add_subplot(3,4,num+1)
    ax.imshow(image, cmap=plt.cm.bone)
    ax.set_title(f"Alter des Patienten:{patient[num].PatientAge}\n Geschlecht: {patient[r
plt.show()
```



In [4]:

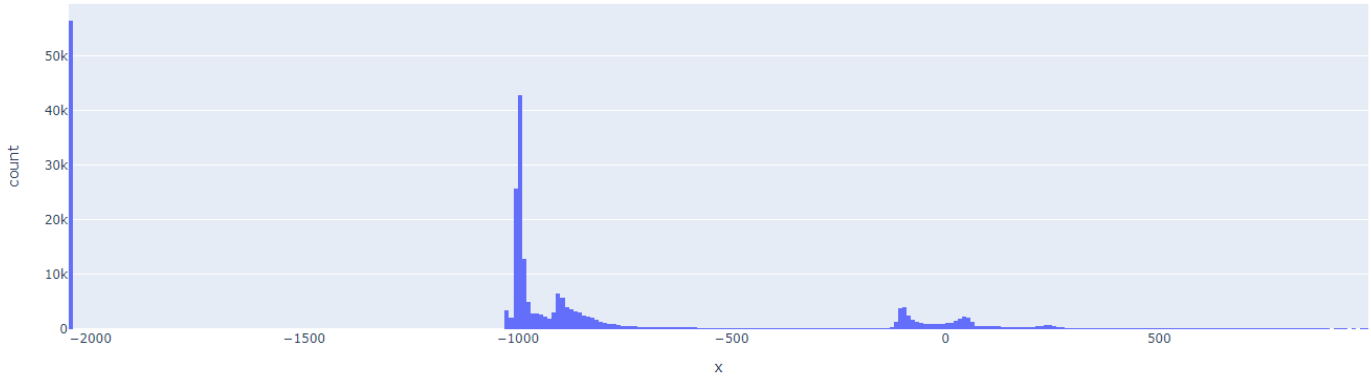
```
print('Dicom ID (0-99):')
simg = wg.BoundedIntText(value=4, min=0, max=99, step=1, disabled=False) #
display(simg)
```

```
Dicom ID (0-99):
```

```
BoundedIntText(value=4, max=99)
```

In [35]:

```
# Anzeigen eines Histogramms der HU
img = np.copy(imgs[simg.value])
fig = px.histogram(x=img.flatten())
fig.show()
```

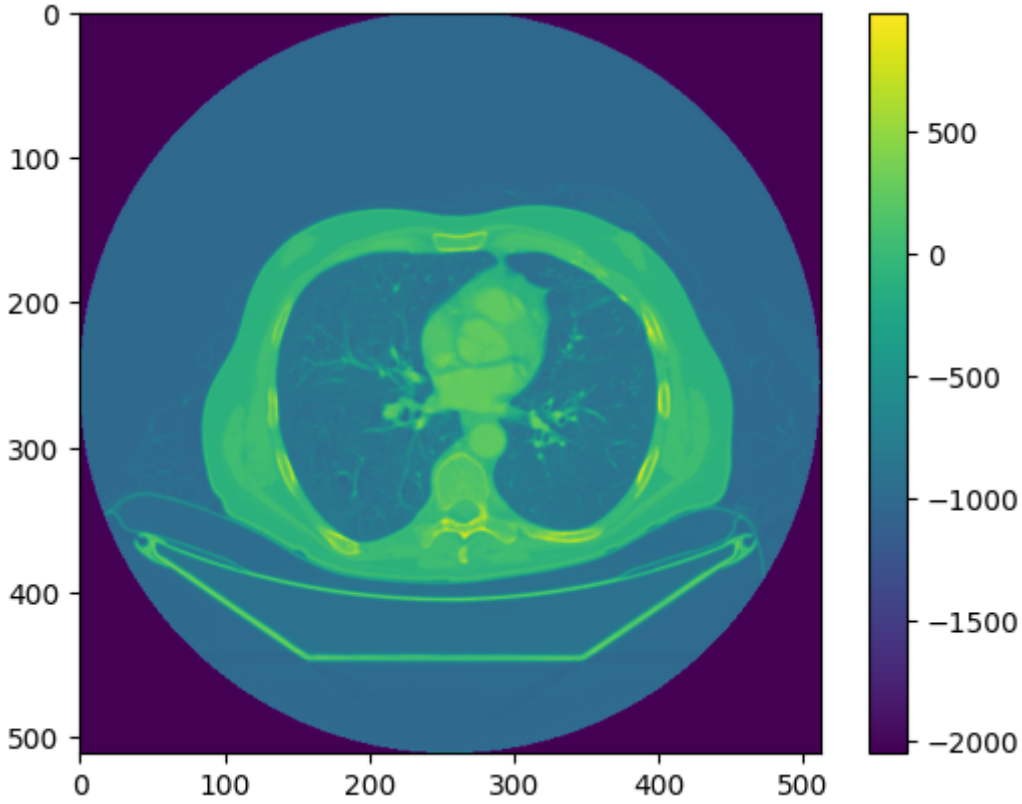


In [6]:

```
# Anzeigen des Bildes mit HU als Pixelwerte
plt.imshow(imgs[simg.value])
plt.colorbar()
```

Out[6]:

```
<matplotlib.colorbar.Colorbar at 0x249b9a80dc0>
```



Entfernen nicht relevanter Bereiche

In [7]:

```
print('Schwellenwert:')
lt = wg.IntSlider(value=-320, min=imgs.min(), max=imgs.max(), step=1, disabled=False)
display(lt)
```

```
Schwellenwert:
```

```
IntSlider(value=-320, max=3071, min=-2048)
```

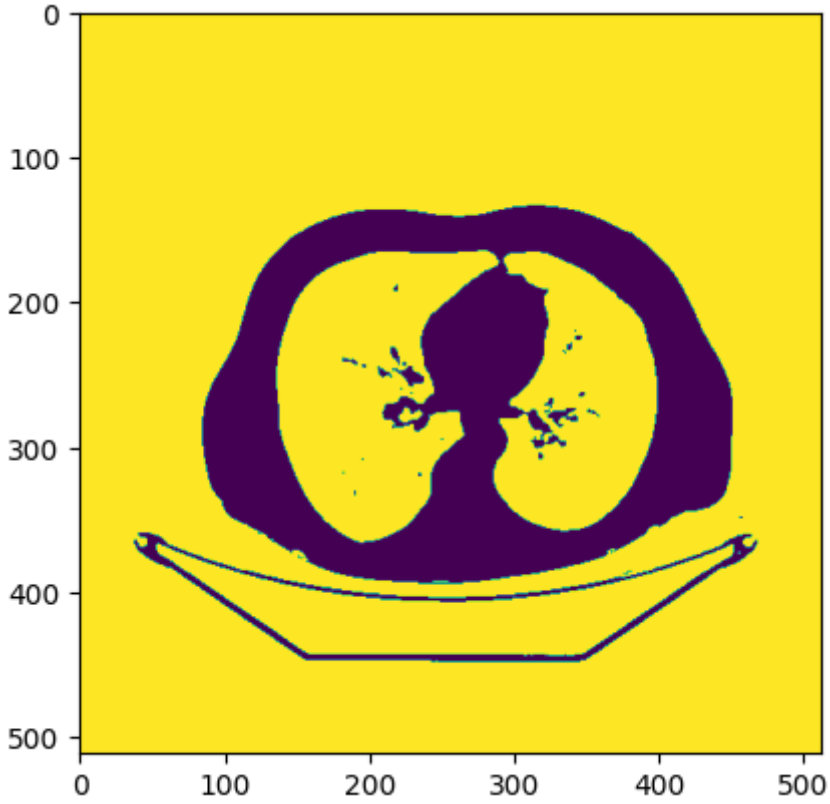
CT-Zahl der Lunge liegt zwischen -550 und -950

In [8]:

```
# Nur Pixel unter dem Schwellenwert anzeigen (Ausgabe als Boolesche Werte)
mask = imgs[simg.value] < lt.value
plt.imshow(mask)
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x249b9452c70>
```

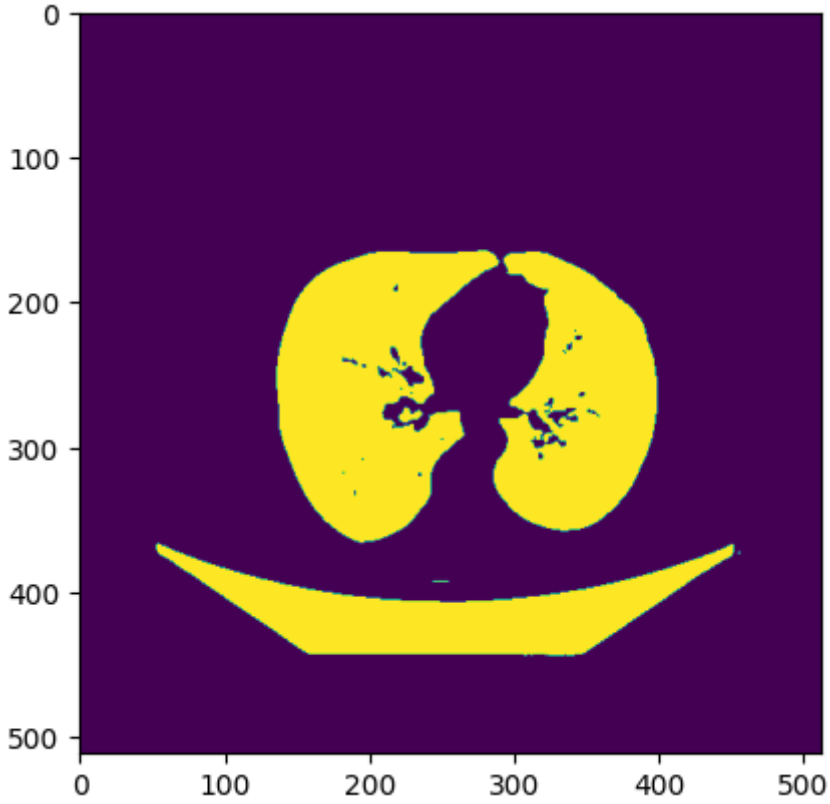


In [9]:

```
# Entfernen der Bereiche am Bildrand
mask = clear_border(mask)
plt.imshow(mask)
```

Out[9]:

```
<matplotlib.image.AxesImage at 0x249b93663d0>
```

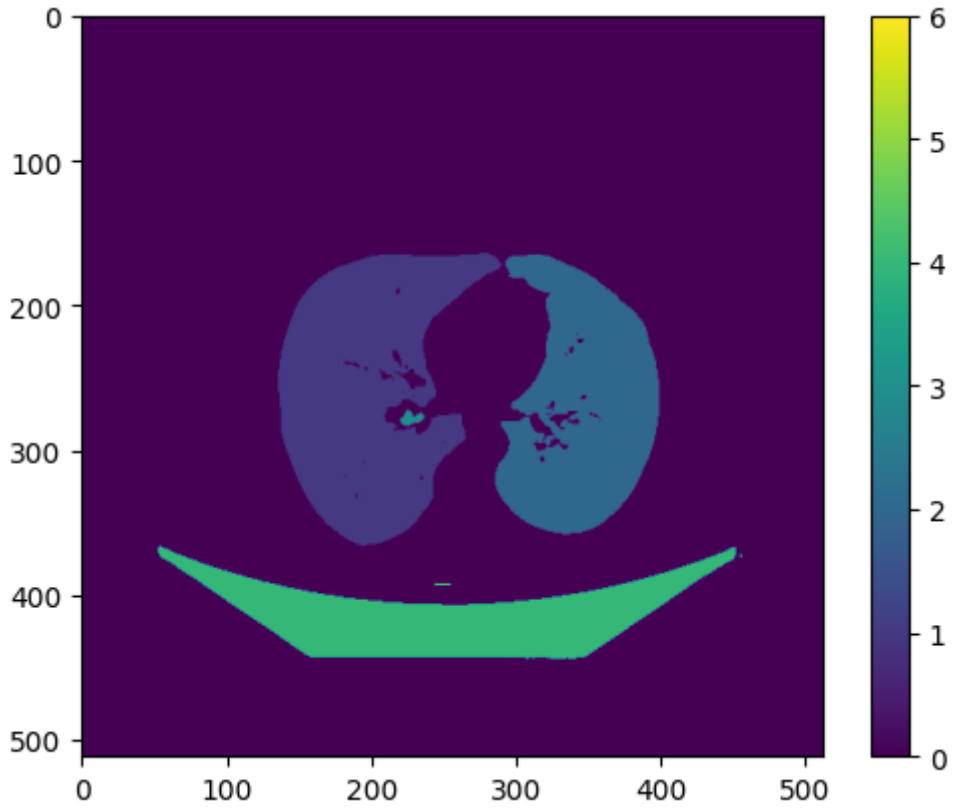


In [10]:

```
# Einzelne zusammenhängende Bereiche jeweils mit einem Label versehen (zur Unterscheidung
mask_labeled = label(mask)
plt.imshow(mask_labeled)
plt.colorbar()
```

Out[10]:

```
<matplotlib.colorbar.Colorbar at 0x249b96ba6d0>
```



Der erste Schritt, um nur noch die Lungenflügel anzuzeigen, ist, die kleinen Bereiche zu entfernen. Aus diesem Grund berechnen wir die Größe der einzelnen Bereiche, sortieren sie der Größe nach und verwerfen alles bis auf die 3 Größten Bereiche. Dass wir 3 Bereiche wählen, obwohl wir nur 2 Lungenflügel darstellen wollen, liegt daran, dass auf manchen Bildern der Tisch größer sein könnte als einer der beiden Flügel. So gehen wir also sicher, dass die interessanten Bereiche auf dem Bild verbleiben.

In [11]:

```
slc = mask_labeled
rps = regionprops(slc)
areas = [r.area for r in rps]
print(len(areas))   # Anzahl der einzelnen Bereiche
areas               # Zeigt die Größe der einzelnen Bereiche
```

```
6
```

Out[11]:

```
[18711.0, 14275.0, 103.0, 12390.0, 2.0, 11.0]
```

In [12]:

```
# Sortiert die Bereiche ihrer Größe entsprechend (groß -> klein)
idxs = np.argsort(areas)[::-1] # -1 kehrt die Reihenfolge um
idxs
```

Out[12]:

```
array([0, 1, 3, 2, 5, 4], dtype=int64)
```

In [13]:

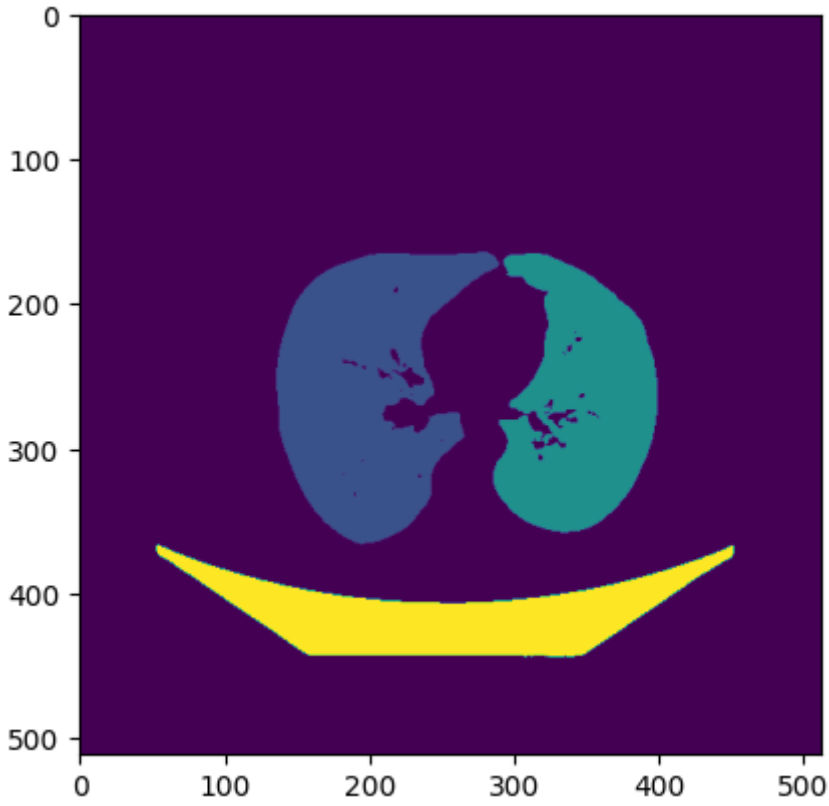
```
# Entfernen aller Bereiche bis auf die 3 größten
new_slc = np.zeros_like(slc)
for i in idxs[:3]:
    new_slc[tuple(rps[i].coords.T)] = i+1
```

In [14]:

```
plt.imshow(new_slc)
```

Out[14]:

```
<matplotlib.image.AxesImage at 0x249b9888640>
```



Um auch noch den Tisch zu entfernen, berechnen wir zunächst die Massenmittelpunkte der einzelnen Bereiche und betrachten deren y-Koordinate.

In [15]:

```
labels = label(new_slc, background=0)
center_of_mass(labels==3) # Gibt die y- und x-Koordinaten des Massenschwerpunkts von Bere
```

Out[15]:

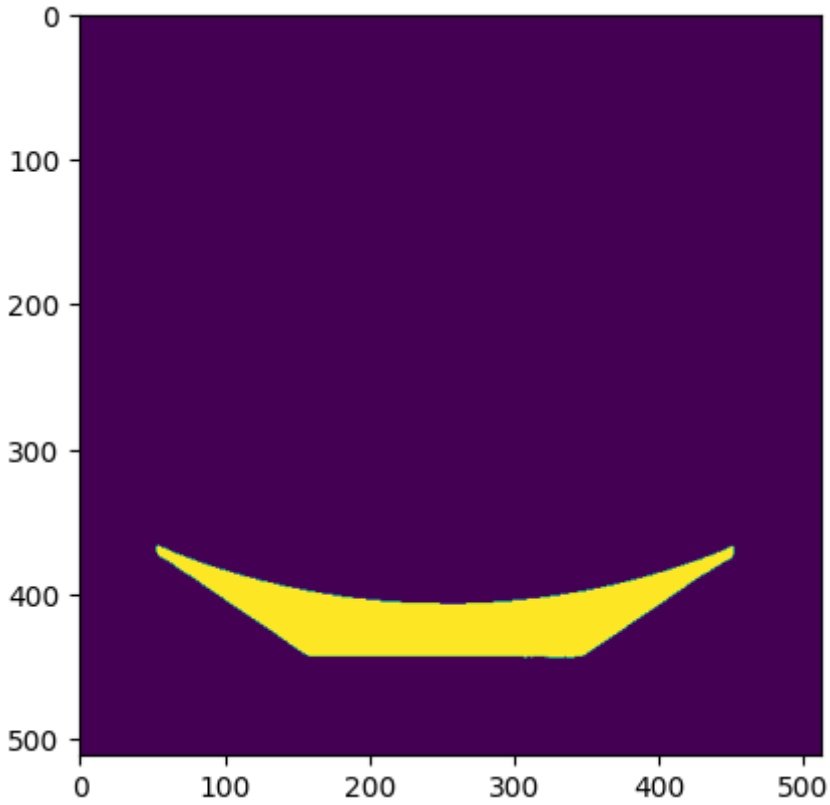
```
(415.72518159806293, 252.47675544794188)
```

In [16]:

```
plt.imshow(labels==3) # Anzeigen des Bereichs mit dem Label 3
```

Out[16]:

```
<matplotlib.image.AxesImage at 0x249b98db310>
```



In [17]:

```
# Definition der Funktion, um den Tisch zu entfernen
def delete_table(slc_ot):
    labels = label(slc_ot, background=0)
    idxs = np.unique(labels)[1:] # Einzelne Bereiche ohne Hintergrund (0 = Hintergund =>
    COM_ys = np.array([center_of_mass(labels==i)[0] for i in idxs]) # Berechnen der y-Koo
    for idx, COM_y in zip(idxs, COM_ys):
        if (COM_y < 0.3*slc_ot.shape[0]):   # Bereiche mit Massenschwerpunkten in den unt
            slc_ot[labels==idx] = 0
        elif (COM_y > 0.6*slc_ot.shape[0]): # Bereiche mit  Massenschwerpunkten in den ob
            slc_ot[labels==idx] = 0
    return slc_ot
```

In [18]:

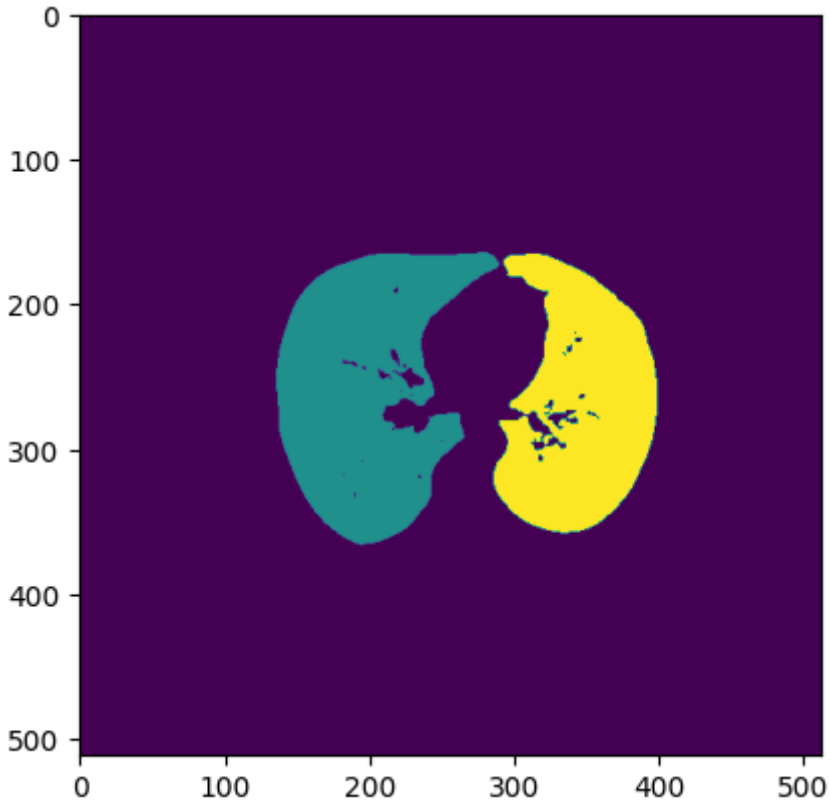
```
mask_ot = delete_table(new_slc) # Funktion ausführen
```

In [19]:

```
plt.imshow(mask_ot)
```

Out[19]:

```
<matplotlib.image.AxesImage at 0x249b93cc460>
```

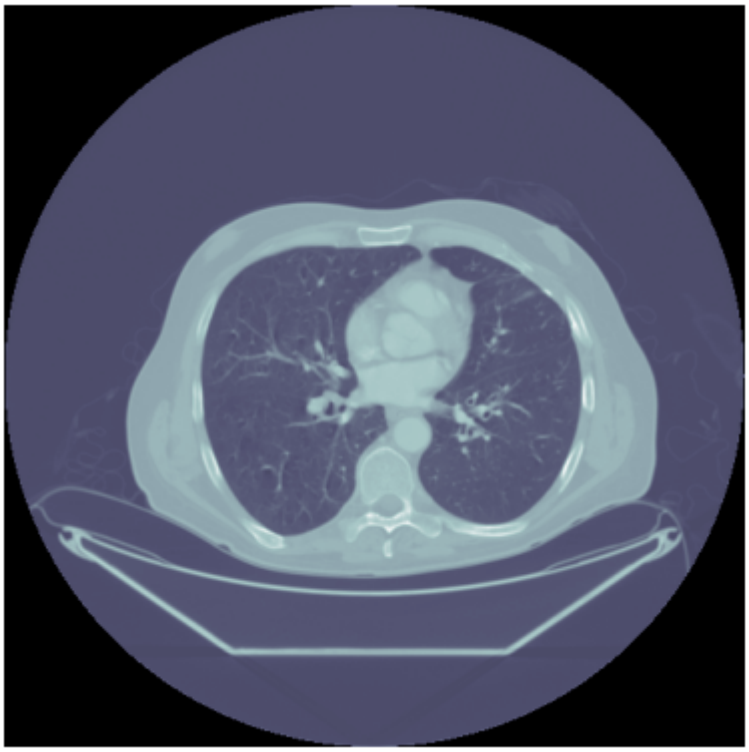


In [20]:

```
# Gibt allen Bereichen den gleichen Wert (gleiche Farbe)
labels = label(mask_ot, background=0)
new_slc[labels==1] = 1
new_slc[labels==2] = 1
plt.imshow(new_slc)
plt.axis('off')
plt.savefig('output/mask.png', transparent=True) # Speichern des Bildes für OpenCV
```

In [21]:

```
# Speichern des Bildes für OpenCV
plt.axis('off')
plt.imshow(imgs[simg.value], cmap=plt.cm.bone)
plt.savefig('output/image.png', transparent=True)
```



Anwenden der Maske auf das Ausgangsbild

In [22]:

```
im_gray = cv2.imread('output/mask.png', cv2.IMREAD_GRAYSCALE) # Maske als Graustufenbild
```

In [23]:

```
(thresh, im_bw) = cv2.threshold(im_gray, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU) #
cv2.imwrite('output/bin_mask.png', im_bw)
```

Out[23]:

```
True
```

In [24]:

```
imags = cv2.imread('output/image.png')
maske = cv2.imread('output/bin_mask.png')
```

In [25]:

```
masked = cv2.bitwise_and(imags,maske) # Binäre Maske auf das Ausgangsbild anwenden
cv2.imwrite('output/masked.png', masked) # Bild speichern
```

Out[25]:

```
True
```

In [26]:

```
# maskiertes Bild anzeigen
plt.axis('off')
plt.imshow(masked)
```

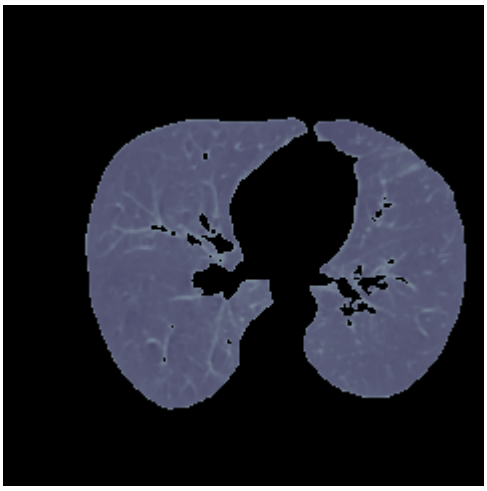
Out[26]:

```
<matplotlib.image.AxesImage at 0x249b92b1ca0>
```

Zuschneiden des maskierten Bildes

In [27]:

```
# Bild öffnen
image = Image.open("output/masked.png")

# Dimensionen des Bildes erhalten
width, height = image.size

# Berechnen der Größe der schwarzen Box
box_size = min(width, height)

# Berechnen der Koordinaten des zentralen Objekts
object_size = box_size // 2
left = (width - object_size) // 2
top = (height - object_size) // 2
right = left + object_size
bottom = top + object_size

# Zuschneiden des Bildes auf das zentrale Objekt
cropped_image = image.crop((left, top, right, bottom))
cropped_image.save("output/output_image.png")
display(cropped_image)
```



Quellen:
https://github.com/lukepolson/youtube_channel/blob/main/Python%20Tutorial%20Series/image_processing1.ipynb
https://www.kaggle.com/code/modoucair/medical-image-analysis
https://towardsdatascience.com/medical-image-pre-processing-with-python-d07694852606
https://dicom.innolitics.com/ciods/digital-x-ray-image/dx-image/00281052
https://flexikon.doccheck.com/de/Hounsfield-Skala
https://www.researchgate.net/figure/Abbildung-2513-Die-Hounsfield-Skala-HU-fuer-verschiedene-Gewebearten-74_fig1_286375965