**ЛАБОРАТОРНАЯ РАБОТА №4.**
**СЕГМЕНТАЦИЯ ИЗОБРАЖЕНИЙ**

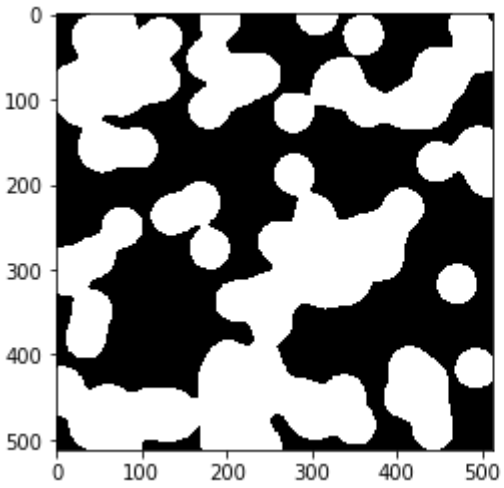
Цель занятия: ознакомление студентов с основными принципами и методами сегментации изображений, разделением изображения на смысловые компоненты, что является важным инструментом в компьютерном зрении и обработке изображений.

Имеются ряд библиотек, написанных для анализа изображений.




Scikit-image — это библиотека Python, предназначенная для обработки изображений. Установка scikit-image устанавливается следующим образом:

```
pip install -U scikit-image(Linux and OSX)
pip install scikit-image(Windows)
# For Conda-based distributions
conda install scikit-image
```




Обзор изображений в Python.

Прежде чем приступить к техническим аспектам сегментации изображений, важно немного ознакомиться с экосистемой изображений Scikit и с тем, как она обрабатывает изображения.

**Импорт GrayScale Image из библиотеки skimage**

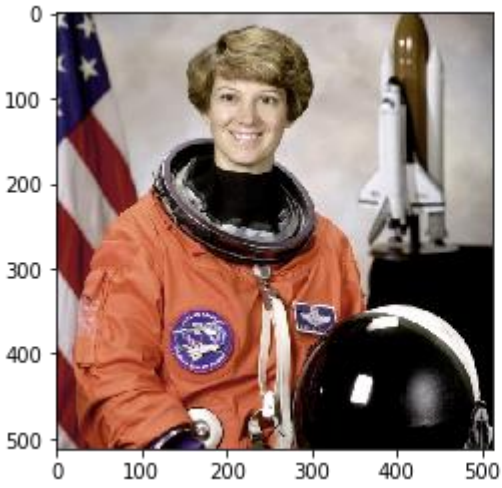
Модуль данных skimage содержит несколько встроенных примеров наборов данных, которые обычно хранятся в формате jpeg или png.

```
from skimage import data
```

```
        import numpy as np
        import matplotlib.pyplot as plt
        image = data.binary_blobs()
        plt.imshow(image, cmap='gray')
```

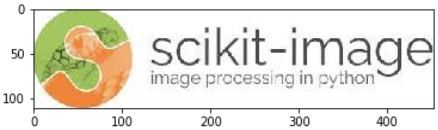
Импорт цветного изображения из библиотеки skimage

```
from skimage import data
        import numpy as np
        import matplotlib.pyplot as plt
        image = data.astronaut()
        plt.imshow(image)
```




Импорт изображения из внешнего источника

```
# The I/O module is used for importing the image
        from skimage import data
        import numpy as np
        import matplotlib.pyplot as plt
        from skimage import io
        image = io.imread('skimage_logo.png')
        plt.imshow(image);
```




Загрузка нескольких изображений

```
images = io.ImageCollection('../images/*.png:../images/*.jpg')
        print('Type:', type(images))
        images.files
```

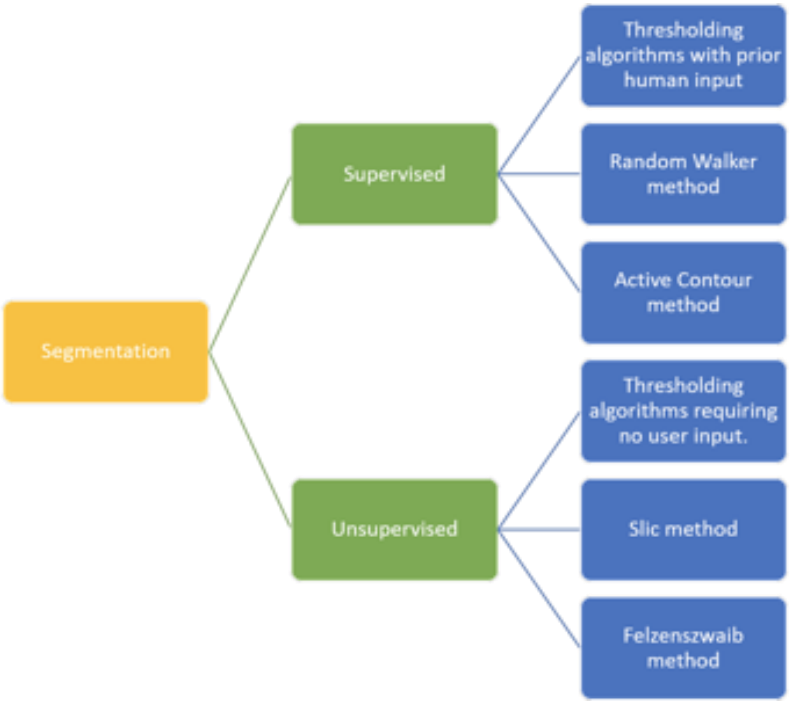
*Out[]: Type: <**class** '**skimage.io.collection.ImageCollection**'>*

Сохранение изображений

```
#Saving file as 'logo.png'
io.imsave('logo.png', logo)
```

**Сегментация изображения**

Теперь, когда у нас есть представление о scikit-image, предлагаем рассмотреть детали сегментации изображений. Сегментация изображения – это процесс разделения цифрового изображения на несколько сегментов, чтобы упростить и / или изменить представление изображения на что-то более значимое и более простое для анализа.




*Некоторые из алгоритмов сегментации доступны в библиотеке scikit-image*

Сегментация с учителем: некоторые предварительные знания, возможно, из человеческого ввода, используются для руководства алгоритмом.
Сегментация без учителем: предварительных знаний не требуется. Эти алгоритмы пытаются автоматически разделить изображения на значимые области. Пользователь по-прежнему может настраивать определенные параметры для получения желаемых результатов.

Обычный импорт

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

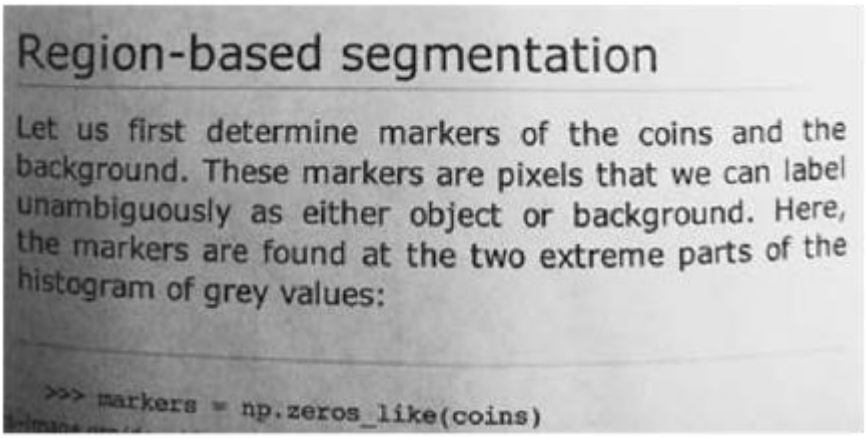
```
import skimage.segmentation as seg
import skimage.filters as filters
import skimage.draw as draw
import skimage.color as color
```

Простая функция для построения изображений

```
def image_show(image, nrows=1, ncols=1, cmap='gray'):
    fig, ax = plt.subplots(nrows=nrows, ncols=ncols, figsize=(14, 14))
    ax.imshow(image, cmap='gray')
    ax.axis('off')
    return fig, ax
```

Образ

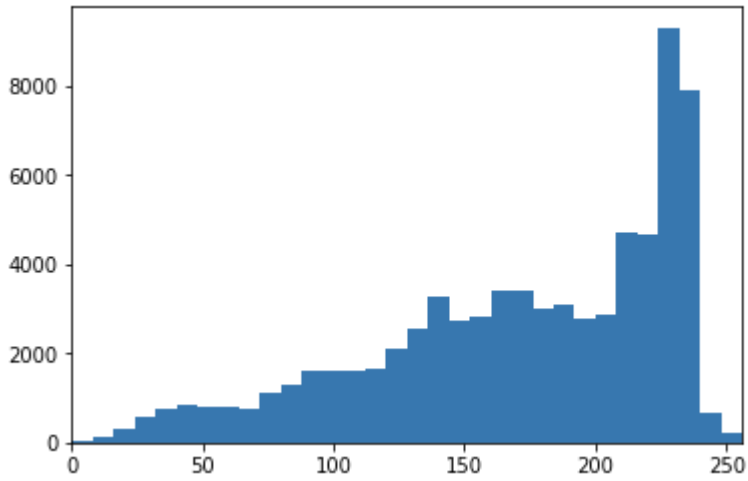
```
text = data.page()
image_show(text)
```




Это изображение немного темнее, но, возможно, мы все же сможем выбрать значение, которое даст нам разумную сегментацию без каких-либо сложных алгоритмов. Теперь, чтобы помочь нам в выборе этого значения, мы будем использовать гистограмму.

В данном случае гистограмма показывает количество пикселей в изображении с различными значениями интенсивности, найденными в этом изображении. Проще говоря, гистограмма — это график, на котором ось X показывает все значения, которые есть на изображении, а ось Y показывает частоту этих значений.

```
fig, ax = plt.subplots(1, 1)
ax.hist(text.ravel(), bins=32, range=[0, 256])
ax.set_xlim(0, 256);
```

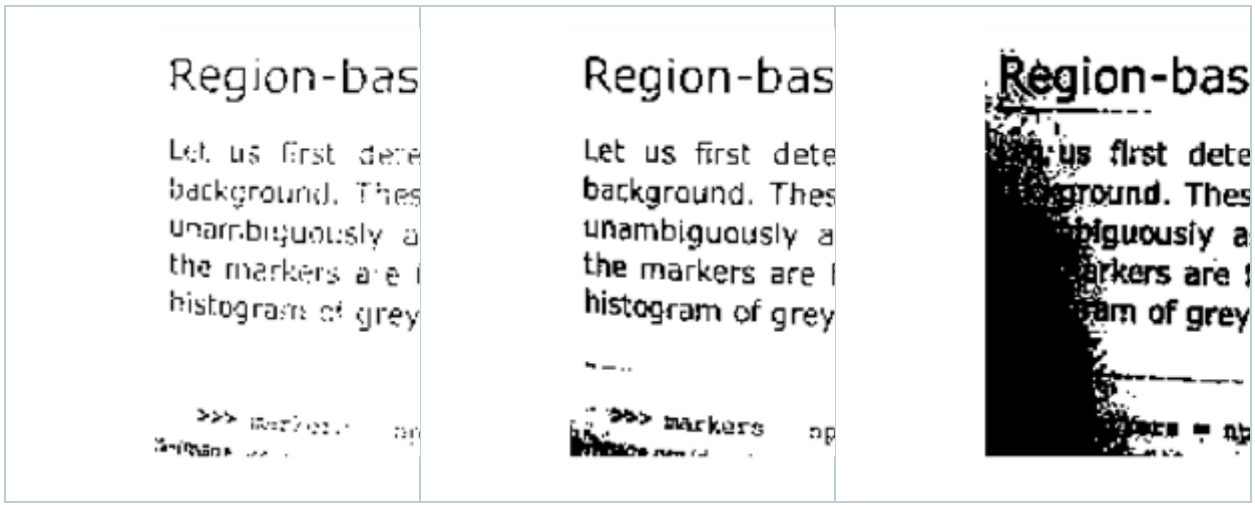



Наш пример оказался 8-битным изображением, поэтому мы имеем 256 возможных значений по оси X. По гистограмме видно, что существует концентрация довольно светлых пикселей (0: черный, 255: белый). Скорее всего, это наш довольно светлый текстовый фон, но остальное немного размыто. Идеальная гистограмма сегментации была бы бимодальной, чтобы мы могли выбрать число прямо посередине. Теперь давайте попробуем создать несколько сегментированных изображений на основе простого порогового значения.

Контролируемый порог

Поскольку мы сами выбираем значение порогового значения, мы называем это контролируемым пороговым значением.

text_segmented = text > (value concluded **from** histogram i.e 50,70,120 )
image_show(text_segmented);



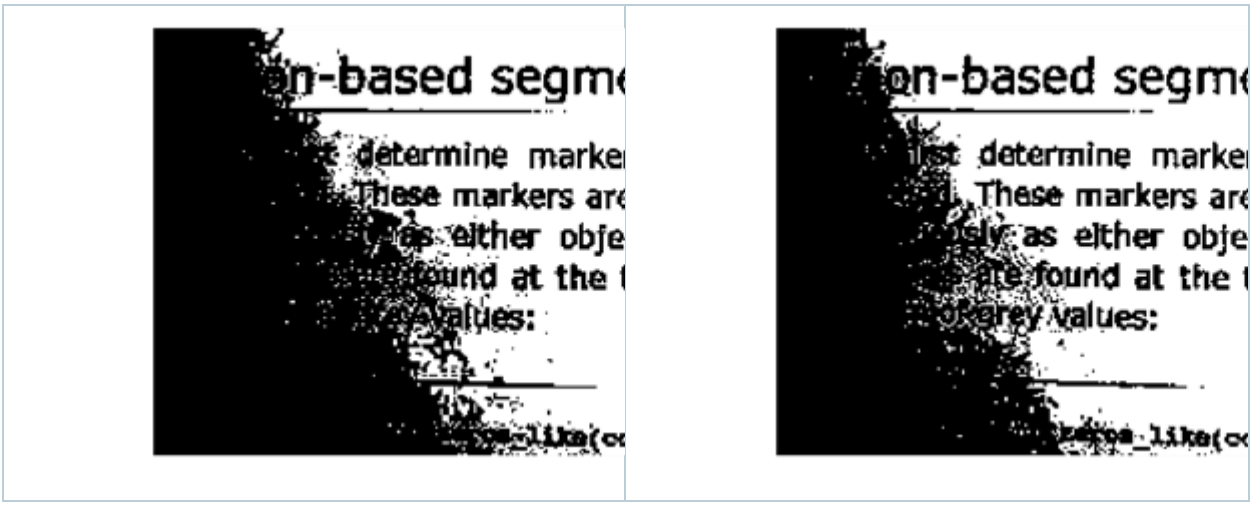

*Слева: текст> 50 | Середина: текст> 70 | Справа: текст> 120*

Мы не получили идеальных результатов, так как тень слева создает проблемы. Давайте попробуем с порогом без присмотра сейчас.

Неконтролируемый порог

Неконтролируемый порог Scikit-image имеет ряд автоматических методов определения порога, которые не требуют ввода при выборе оптимального порога. Вот некоторые из методов: otsu, li, local.

```
text_threshold = filters.threshold_  # Hit tab with the cursor after the underscore to get all the methods.
image_show(text < text_threshold);
```



*Слева otsu || Справа: li*

В случае local, нам также нужно указать block_size. Offset помогает настроить изображение для лучших результатов.

```
text_threshold = filters.threshold_local(text,block_size=51, offset=10)
image_show(text > text_threshold);
```

## Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```