Теоретический материал курса "Работа с файлами и изображениями на Python"

Введение

Графический формат изображения – это способ записи графической информации. Графические форматы файлов предназначены для хранения изображений.

Графические файлы имеют заголовки, которые являются частью файла, в них записана необходимая для работы с файлом информация (тип, размер, расположение данных в файле и другое).

Само видимое изображение представляет собой набор (массив) пикселей. Пиксель – это строительный блок изображения. Изображение можно представить в виде сетки, каждый квадрат в этой сетке содержит один пиксель и имеет свои координаты. Координата (0,0) соответствует пикселю в верхнем левом углу изображения.

Если у нас есть изображение, состоящее из 500 строк и 600 столбцов (500x600), то изображение будет содержать 500*600 = 300’000 пикселей

Пиксели могут быть представлены в различном формате, наиболее распространенным для цветных изображений является RGB (RED, GREEN, BLUE) – это цветовая модель, позволяющая задавать цвет с помощью численных значений трех составляющих. Одно значение для красной компоненты, одно для зеленой и одно для синей. Каждая компонента кодируется числом от 0 до 255, то есть для ее хранения нужно 8 бит – 1 байт. Например, пиксель со значением (0, 0, 0) будет черного цвета, (255, 0, 0) – красного, (255, 255, 255) – белого.

Также есть расширение модели RGB – RGBA, тогда для каждого пикселя, помимо цветовых каналов добавляется альфа-канал (Alpha), он отвечает за прозрачность пикселя. Для полной непрозрачности нужно использовать значение 255, для полной прозрачности – 0.

Если изображение представлено в оттенках серого, то каждый пиксель может кодироваться одним числом от 0 до 255. 0 – черный цвет, 255 – белый цвет. Остальные значения соответствуют различным оттенкам серого, от светлых к темным.

Существуют и другие модели цвета, такие как CMYK (Cyan, Magenta, Yellow, Key/Black), которая используется для печати, и HSL(Hue, Saturation, Lightness), которая позволяет задавать цвета с помощью оттенка, насыщенности и яркости.

Графические форматы могут быть сжатыми или несжатыми. Несжатые форматы, например BMP, сохраняют изображения без потерь качества, но занимают больше места на диске. Сжатые форматы, такие как JPEG или PNG, используют алгоритмы сжатия, что позволяет сохранить место на диске, но сжатие может привести к потере качества изображения.

Библиотека Pillow -> Базовые функции

Описание библиотеки

Работать с бинарными файлами изображений неудобно, для выполнения простейшего действия с изображениями необходимо написать много кода.

Поэтому в Python существует множество библиотек, которые сильно упрощают работу с изображениями. Библиотеки предоставляют структуры данных для представления изображений, преобразуют данные изображения в удобный для работы с ними вид.

Библиотека Pillow – это мощный инструмент языка Python для работы с изображениями. Данная библиотека умеет работать с различными форматами изображений, например, такими как: JPEG, PNG, BMP.

Ссылка на официальную документацию:
https://pillow.readthedocs.io/en/stable/reference/index.html#

Для установки данной библиотеки можно использовать утилиту pip:

pip install pillow

Имя библиотеки в Python - PIL.

Базовые функции

В Pillow основным модулем для работы с изображениями является Image. С помощью него можно считывать изображение и работать с ним.

Модуль Image необходимо импортировать из библиотеки:

from PIL import Image

Для открытия изображения используется метод open модуля Image. Посмотреть изображение можно с помощью метода show, этот метод сохраняет изображение как временный файл и отображает его, с помощью ПО операционной системы. Для закрытия файла используется метод close. См. рисунок.

```
img = Image.open('jpg.jpg')

img.show()

img.close()
```

Также после манипуляций с изображением, его можно сохранить, используя метод save:

img.save('./images/img.png')

С помощью поля format можно узнать формат изображения, с помощью size – его ширину и высоту в пикселах. Также ширину и высоту можно узнать с помощью полей width и height соответственно:

```
print('image size: ', img.size)
print('image format: ', img.format)
print('image height: ', img.height)
print('image width: ', img.width)
```

```
image size:  (183, 275)
image format:  PNG
image height:  275
image width:  183
```

Представление в виде массива

С помощью модуля numpy можно получить представление изображения попиксельно в виде numpy массива. Для этого нужно изображение типа Image обернуть в numpy.array массив.

```
arr = np.array(img)
print(arr)
```

```
[[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [238 238 238]
  [238 238 238]
  [238 238 238]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [238 238 238]
  [238 238 238]
  [238 238 238]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [238 238 238]
  [238 238 238]
  [238 238 238]]

 ...

 [[255 255 255]
```

numpy.array(img)

Внешний массив содержит внутренние массивы строк, которые в свою очередь содержат массивы пикселей. В данном случае, каждый пиксель представляет собой массив из 3-х чисел – это значения в RGB модели.

Рассмотрим, как можно перекрасить, например, все белые пиксели изображения (255, 255, 255) в красные (255, 0, 0):

```
arr = np.array(img)
for line in arr:
    for pixel_index in range(len(line)):
        if line[pixel_index][0] == 255 and line[pixel_index][1] == 255 and line[pixel_index][2]== 255:
            line[pixel_index] = np.array([255,0,0])

img_result = Image.fromarray(arr)
```

Здесь, для преобразования Numpy массива в изображения используется метод Image.fromarray().

Работа с пикселями

Также работать с пикселями можно с помощью методов getpixel (получение пикселя по координатам) и putpixel(вставка пикселя по координатам).

- getpixel() принимает на вход 1 аргумент – координаты пикселя (x,y) и возвращает значение этого пикселя.
- putpixel() принимает на вход 2 аргумента – координаты пикселя (x,y) и цвет , в который этот пиксель нужно закрасить.

Пример кода, закрашивающий первые 100 строк изображения в красный цвет:

```
for y in range(100):
    for x in range(img.width):
        img.putpixel((x,y), (255,0,0))
```

Новое изображение

С помощью библиотеки Pillow можно создавать свои изображения, для этого используется метод new.

Его можно использовать, например, следующим образом:

```
width=300
height = 200
img = Image.new(mode='RGB',size = (width,height), color = 'pink')
img
```


- mode – тип изображения, в данном случае ‘RGB’. Также могут быть варианты ‘1’ – 1-битовое черное-белое изображение; ‘P’ – использование палитры цветов; ‘RGBA’ – с альфа-каналом; и другие.
- size – размер изображения
- color – цвет изображения

Библиотека Pillow -> Геометрия

Отображение геометрических фигур с помощью библиотеки Pillow

Для рисования различных геометрических фигур на изображении используется модуль ImageDraw библиотеки Pillow. Его можно подключить с помощью следующего кода:

from PIL import ImageDraw

Документация к модулю ImageDraw - https://pillow.readthedocs.io/en/stable/reference/ImageDraw.html

Модуль ImageDraw предоставляет возможность добавлять на изображение различную 2D-графику.

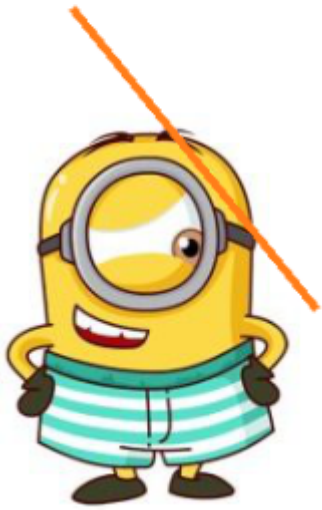
Для того, чтобы начать рисовать на изображении, нужно передать методу ImageDraw.Draw Pillow-изображение (PIL.Image.Image). Данный метод вернет объект ImageDraw, с помощью которого можно будет изменять переданное изображение. Этот объект предоставляет методы для рисования различных фигур, текста и других элементов на изображении.

Рассмотрим методы, с помощью которых можно рисовать геометрические фигуры:

- Метод line позволяет нарисовать прямую линию. Первым аргументом принимает tuple с координатами начала и конца линии (x1,y1,x2,y2). Также можно передать аргументы width – ширина линии и fill – цвет линии. Цвет можно передать как tuple (r,g,b) или с помощью названия какого-либо стандартного цвета в виде строки (‘blue’, ‘red’, ‘yellow’ и т.д.)

Пример:

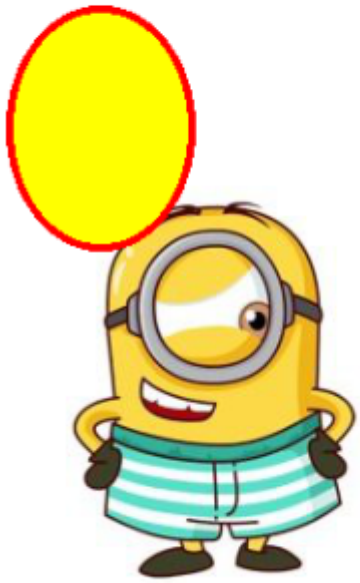
```
draw = ImageDraw.Draw(img)
draw.line((70,40,200,200),width=4,fill=(255,125,25))
img
```



Также, с помощью метода line можно нарисовать сразу несколько прямых линий, соединенных между собой. Для этого в качестве первого аргумента можно передать вместо 2-х координат точек, любое другое большее количество, тогда нарисуется ломаная, соединяющая все точки последовательно.

- Метод ellipse позволяет рисовать эллипс. Первым аргументом принимает верхнюю левую и правую нижнюю координату описывающего прямоугольника (x1, y1, x2, y2). Также можно передать аргументы fill – цвет заливки эллипса, outline – цвет линии обводки, width – ширина линии обводки.

```
img = Image.open("min.jpg")
draw = ImageDraw.Draw(img)
draw.ellipse((0,0,100,130),fill='yellow',width=4,outline='red')
img
```

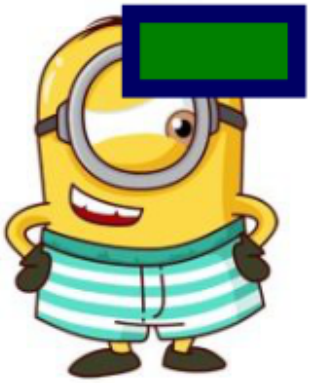


Пример:

- Метод rectangle позволяет рисовать прямоугольник. Первым аргумент принимает tuple с верхней левой и нижней правой координатой прямоугольник (x1,y1,x2,y2). Также можно передать аргументы fill – цвет заливки прямоугольника, outline – цвет линии обводки, width – ширина линии обводки.

Пример:

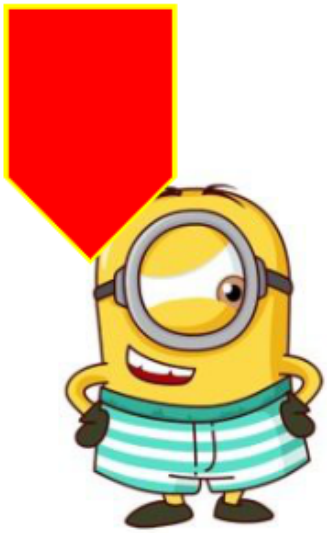
```
draw = ImageDraw.Draw(img)
draw.rectangle((100,100,200,150),fill='green',outline=(0,0,100),width=10)
img
```



- Метод polygon позволяет рисовать многоугольники. Первым аргументом передается набор координат – точек многоугольника, которые будут последовательно соединяться. Также можно передать аргументы fill – цвет заливки многоугольника, outline – цвет линии обводки, width – ширина линии обводки.

Пример:

```
draw = ImageDraw.Draw(img)
draw.polygon((0,0,0,100,50,150,100,100,100,0),width=3,fill='red',outline='yellow')
img
```



- Метод point позволяет рисовать точки. С помощью данного метода можно изменять цвет множества пикселей, координаты которых передаются. В аргументе fill передается цвет пикселей.

Пример использования:

draw.point((0,0,0,100,50,150,100,100,100,0),fill='red')

Такой же результат можно получить вызовом:

draw.point(((0,0),(0,100),(50,150),(100,100),(100,0)),fill='red')

Библиотека Pillow -> Работа с цветом

Для получения пикселе и изменения пикселей в Pillow используются методы getpixel() и putpixel().

- getpixel() принимает на вход 1 аргумент – координаты пикселя (x,y) и возвращает значение этого пикселя.
- putpixel() принимает на вход 2 аргумента – координаты пикселя (x,y) и цвет , в который этот пиксель нужно закрасить.

Пиксель представляет из себя tuple из нескольких элементов. Например, для RGB-формата изображения - это 3 элемента, для RGBA - 4 элемента.

Для оптимизации памяти в изображениях может использоваться так называемая палитра цветов. Если цветов в изображении немного, то вместо того, чтобы для каждого пикселя хранить 3 элемента (для RGB-формата), можно хранить для каждого пикселя индекс в палитре цветов - одно число. Этот индекс будет указывать на цвет в палитре.

С помощью Pillow можно легко получить палитру цветов изображения, если она присутствует.

Посмотрим, как выглядит numpy-массив пикселей изображения, если в нем используется палитра цветов:

```
np.array(img)
```

```
array([[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]], dtype=uint8)
```

Здесь, вместо наборов из 3-х чисел, для каждого пикселя хранится одно число - индекс цвета в палитре.

Получить палитру можно с использованием метода getpalette():

```
print(img.getpalette())
```

```
[0, 0, 0, 255, 255, 255, 235, 235, 235, 24
08, 208, 208, 193, 193, 193, 221, 221, 221
5, 111, 111, 111, 143, 143, 143, 79, 79, 7
62, 62, 62, 130, 130, 130, 165, 165, 165,
178, 178, 90, 90, 90, 101, 101, 101, 39, 3
1, 50, 50, 50, 26, 26, 26, 117, 117, 117,
43, 44, 44, 44, 45, 45, 45, 46, 46, 46, 47
2, 52, 52, 53, 53, 53, 54, 54, 54, 55, 55,
```

В палитре цветов цвета обычно располагаются от наиболее часто встречающихся к наименее встречающимся. Так, в данном случае, под индексом 0 находится черный цвет (0,0,0). Под индексом 1 - белый цвет (255,255,255) и так далее.

Выше - палитра следующего изображения:

Палитру можно изменить с помощью метода putpalette(), передав в него новую палитру. Заменим у этого изображения второй цвет палитры на красный:

```
palette = img.getpalette()
palette[3:6] = 255,0,0
img.putpalette(palette)
```

В итоге получим следующее изображение:

Поле mode у изображения, использующего палитру, будет хранить строку "P". (У RGB изображения - "RGB", у RGBA изображения - "RGBA")

Библиотека Pillow -> Работа с областями изображений, работа с текстом №1

Работа с областями изображения. Pillow.

Рассмотрим способы взаимодействия с областями изображений с помощью библиотеки Pillow. Методы Pillow:

- crop – с помощью данного метода можно обрезать изображение. Обрезается прямоугольная область. В аргументе передается tuple с 4 числами (x1, y1, x2, y2). x1,y1 – координаты верхнего левого угла области; x2,y2 – координаты нижнего правого угла области. Не изменяет текущее изображение, а возвращает новое.

Пример кода, обрезающий нижнюю половину картинки (с округлением вниз при дробном результате):

```
x1 = 0
y1 = img.height//2
x2 = img.width
y2 = img.height
crop_img = img.crop((x1,y1,x2,y2))
```

- resize – данный метод позволяет изменить размер изображения в пикселях. Принимает на вход размер изображения (width, height). Не изменяет текущее изображение, а возвращает новое.

Пример кода, изменяющий высоту изображения, увеличивая ее в 2 раза:

```
resize_img = img.resize((img.width,img.height*2))
```

- reduce – этот метод также позволяет уменьшить размер изображения. Он принимает 1 аргумент – коэффициент уменьшения. Не изменяет текущее изображение, а возвращает новое.
- transpose – данный метод позволяет поворачивать или отображать изображение. Принимает на вход константу из Image, которая определяет, как будет модифицировано изображение. Не изменяет текущее изображение, а возвращает новое.

Пример отзеркаливания изображения по вертикальной оси:

```
img.transpose(Image.FLIP_LEFT_RIGHT)
```

До:

После:

Для отображения по горизонтальной оси можно использовать следующий код:

```
img.transpose(Image.FLIP_TOP_BOTTOM)
```

Для поворота изображения на 90 градусов против часовой стрелки:

```
img.transpose(Image.ROTATE_90)
```

Все возможные константы для этого метода приведены в документации:
https://pillow.readthedocs.io/en/stable/reference/Image.html#PIL.Image.Image.transpose

- rotate – с помощью данного метода можно поворачивать изображение на любое количество градусов. Не изменяет текущее изображение, а возвращает новое.

Пример поворота изображения на 60 градусов против часовой стрелки:

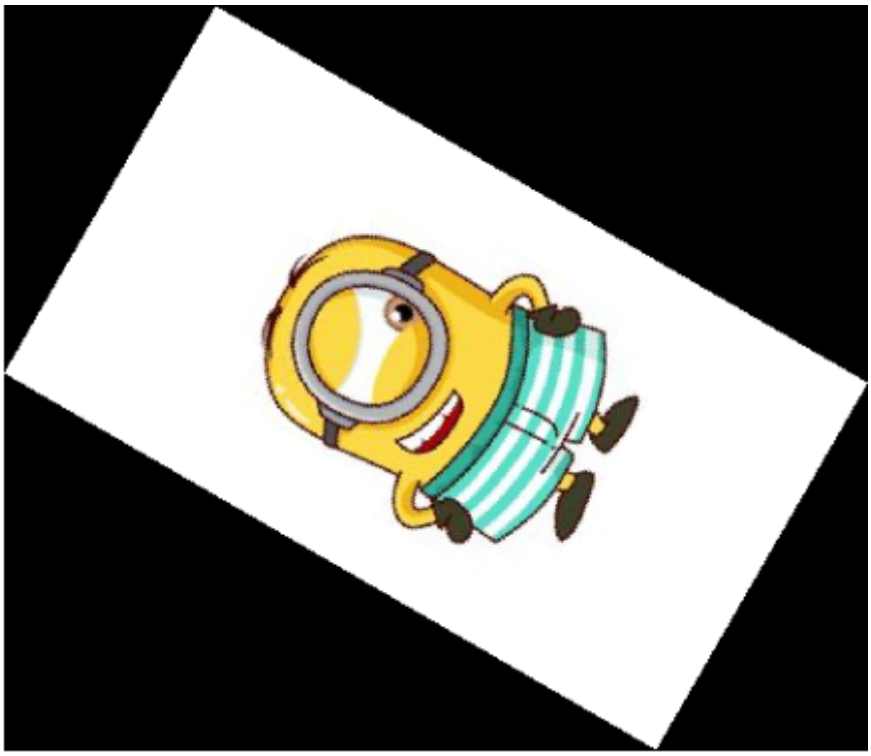
```
Img.rotate(60)
```

До:

После:  По умолчанию изображение возвращается того же размера, что и было, поэтому в данном случае углы картинки не попали в итоговое изображение. Такое поведение можно изменить, и сделать так, чтобы размер итогового изображения увеличился таким образом, чтобы всё перевернутое изображение влезло. Это осуществляется с помощью аргумента expand:

```
img.rotate(60,expand=True)
```

Тогда получим следующее изображение:



- paste – с помощью данного метода можно вставить одно изображение в другое. Если изображение не будет помещаться, то оно автоматически обрежется. Первым аргументом указывается изображение, которое нужно вставить в текущее, втором – координата начала вставки.

Пример:

```
|: img1 = Image.open("jpg.jpg")
   img2 = Image.open("lion.png")
   img1.paste(img2,(100,100))
   img1
```

|:

Библиотека Pillow -> Работа с областями изображений, работа с текстом №2

Работа с текстом. Pillow.

Также с помощью библиотеки Pillow можно вставлять текст в картинку. Это делается с помощью метода text класса Draw. В качестве аргументов передаются координаты начала (x,y), текст – строка, fill – цвет текста, font - шрифт.

В качестве шрифта font можно передать объект шрифта. Он создается с помощью модуля ImageFont библиотеки Pillow, там указывается и размер шрифта.

Пример использования:

```
draw = ImageDraw.Draw(img)
font = ImageFont.truetype('arial',100)
draw.text((200,200),'JPEG',fill='red',font=font)
img
```

Библиотека OpenCV -> Базовые функции

Описание библиотеки

OpenCV - это мощная библиотека компьютерного зрения, с помощью которой можно проводить анализ, классификацию и обработку изображений.

Ссылка на документацию: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

Установить данную библиотеку можно с помощью утилиты pip :

```
pip install opencv-python
```

Рассмотрим возможности данной библиотеки для обработки изображений.

Для подключения библиотеки в свой код можно использовать выражение:

```
import cv2
```

Для чтения изображения с помощью библиотеки OpenCV используется метод imread (cv2.imread(path)), которому передается путь к изображению.

Изображения в OpenCV представляют из себя numpy-массивы (numpy.ndarray).

Отобразить изображение можно с помощью метода imshow. В метод нужно передать название окна, в котором отобразится изображение и само изображение.

После вызова imshow нужно вызвать метод waitKey и передать ему в качестве аргумента число 0. Если этого не сделать, то изображение закроется сразу, после открытия. Чтобы программа продолжила работу нужно закрыть окно изображения.

Сохранить изображение можно с помощью метода imwrite, которому передается путь сохранения и само изображение.

Пример использования данных методов:

```
import cv2 #подклчаем библиотеку
img = cv2.imread('img.jpg') #читаем изображение
cv2.imshow("OpenCV image",img) #отображаем изображение
cv2.waitKey(0) #ждем закрытия окна с изображением
cv2.imwrite('saved_img.jpg',img) #записываем изображение
```

Высоту и ширину изображения можно достать из поля shape изображения:

```
print(img.shape[0])#высота
print(img.shape[1])#ширина
```

Представление в виде массива

Прочитанное изображение можно вывести в виде многомерного numpy-массива. Данный массив содержит строки пикселей. Пиксель может представлять из себя одномерный массив (например, в RGB-изображении), может представлять из себя одно число (например, в черно-белом изображении) .

Пример массива цветного изображения:

```
print(cv2.imread("img.png"))
```

```
[[[  0   0 255]
  [  0   0 255]
  [  0   0 255]
  ...
  [  0   0 255]
  [  0   0 255]
  [  0   0 255]]

 [[  0   0 255]
  [  0   0 255]
  [  0   0 255]
  ...
  [  0   0 255]
  [  0   0 255]
  [  0   0 255]]

 [[  0   0 255]
  [  0   0 255]
  [  0   0 255]
  ...
```

Пример массива черно-белого изображения:

```
[[213 213 213 ... 222 222 222]
 [214 214 214 ... 223 223 223]
 [214 214 214 ... 223 223 223]
 ...
 [213 212 212 ... 205 206 206]
 [213 212 212 ... 205 206 206]
 [213 212 212 ... 204 205 205]]
```

Работа с пикселями

Для доступа к пикселям изображения можно использовать оператор []. При использовании RGB модели, пиксели буду представлены в виде кортежа (b, g, r). Важно! Значения идут в обратном порядке, не (r,g,b).

Распечатаем пиксели второй строки изображения:

```
for col in range(img.shape[1]):
    (b,g,r) = img[1,col]
    print('red:',r,' green:', g, ' blue:', b)
```

```
red: 26  green: 77  blue: 137
red: 26  green: 77  blue: 137
red: 26  green: 77  blue: 137
red: 26  green: 77  blue: 137
red: 26  green: 77  blue: 137
red: 26  green: 77  blue: 137
red: 26  green: 77  blue: 137
red: 26  green: 77  blue: 137
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
red: 55  green: 99  blue: 153
```

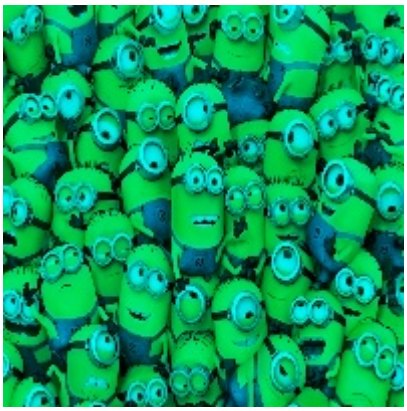
Пикселям можно присваивать значения изменяя значения массива.

Установим значение красной компоненты каждого пикселя изображения в 0 и посмотрим результат:

```
for row in range(img.shape[0]):
    for col in range(img.shape[1]):
        (b,g,r) = img[row,col]
        img[row,col] = (b,g,0)
cv2.imwrite("img_without_red.jpg",img)
```

До:

После:



Новое изображение

Для создания нового изображения нужно воспользоваться функцией numpy - full.

Пример кода, создающего RGB-изображение с размером 300 (высота) на 400 (ширина) пикселей, с цветом (255,100,100):

```
img = numpyp.full((300, 400, 3), [255,100,100], dtype=nunpy.uint8)
```

Тип данных numpy.uint8 означает, что значения пикселей будут представлены целыми числами от 0 до 255.

Библиотека OpenCV -> Геометрия

Отображение геометрических фигур с помощью библиотеки OpenCV

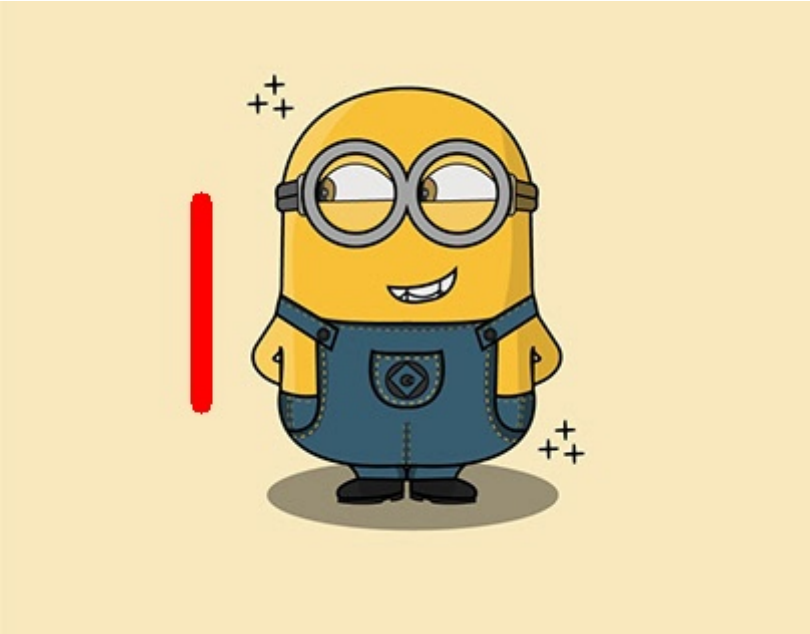
Рассмотрим как можно рисовать геометрические фигуры на изображении с помощью библиотеки OpenCV.

- Рисование линий реализуется методом line, который принимает на вход изображение - image; начальную и конечную координаты линии – start_point, end_point; цвет – color; ширину линии – thickness. Данный метод изменяет переданное изображение.

Пример использования:

```
img = cv2.imread("img.jpeg")
start_point = (100,100)
end_point = (100,200)
color = (0,0,255)
thickness = 10
cv2.line(img, start_point, end_point, color, thickness)
cv2.imwrite("saved.jpeg",img)
```

В данном случае нарисуется красная вертикальная линия, шириной 10.

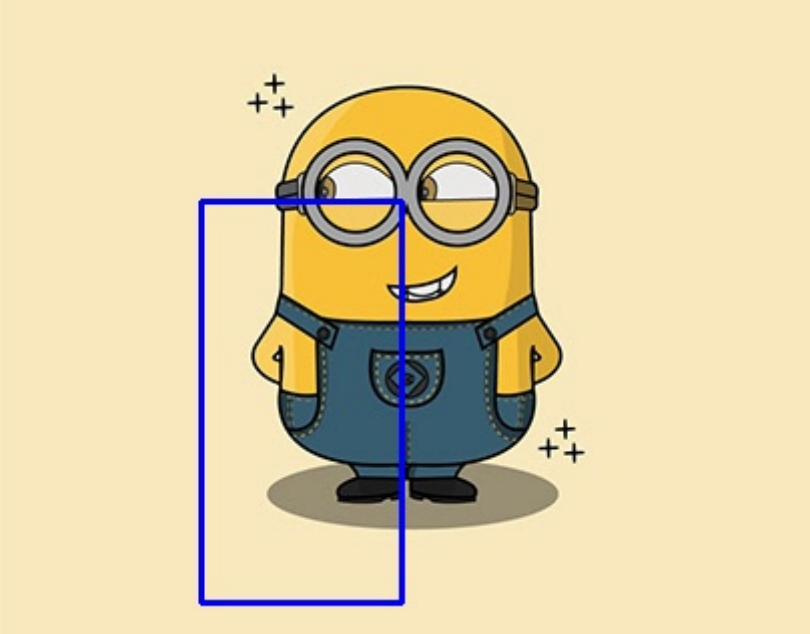


- Рисование прямоугольника реализуется методом rectangle, который принимает на вход изображение - image, координаты верхнего левого угла – start_point, координаты нижнего правого угла – end_point, цвет - color, толщину линии - thickness. Данный метод изменяет переданное изображение.

Если в качестве thickness передать -1, то прямоугольник будет залит указанным цветом.

Пример кода для рисования прямоугольника:

```
img = cv2.imread("img.jpeg")
start_point = (100,100)
end_point = (200,300)
color = (255,0,0)
thickness = 2
cv2.rectangle(img, start_point, end_point, color, thickness)
cv2.imwrite("saved.jpeg",img)
```

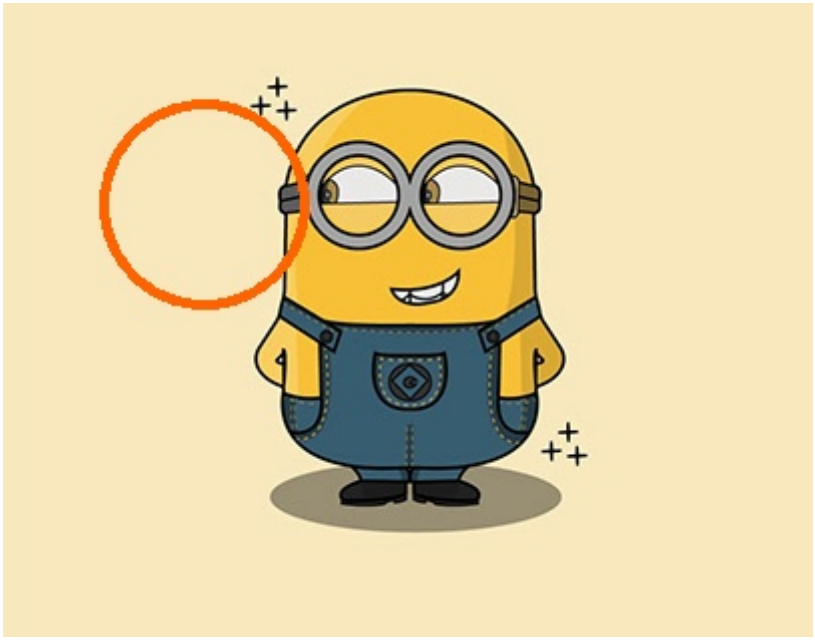


- Для отрисовки окружности можно использовать метод circle. Он принимает на вход изображение – image, координату центра окружности – center_coordinates, радиус окружности – radius, цвет окружности – color, толщину линии – thickness. Данный метод изменяет переданное изображение.

Если в качестве thickness передать -1, то окружность будет залита указанным цветом.

Пример кода для рисования окружности:

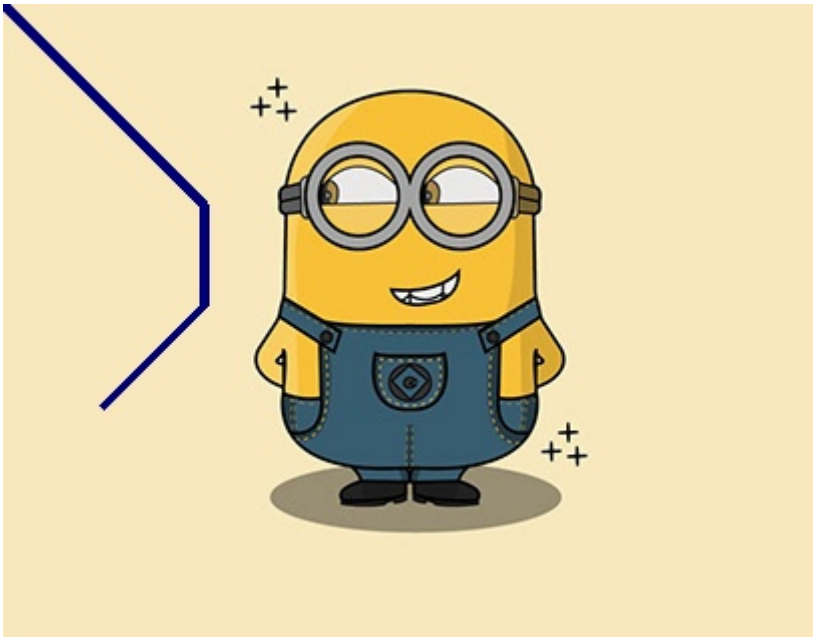
```
img = cv2.imread('img.jpeg')
center_coordinates = (100,100)
radius = 50
color = (0,100,255)
thickness = 3
cv2.circle(img, center_coordinates, radius, color, thickness)
cv2.imwrite("saved.jpeg",img)
```



- Чтобы нарисовать несколько соединенных линий используется метод polylines. Он принимает на вход изображение – image; набор точек, которые будут последовательно соединены – [pts]; флаг того, нужно ли соединять последнюю точку с первой – isClosed; цвет – color; толщина линии -thickness. Данный метод изменяет переданное изображение.

Пример кода для рисования соединенных линий:

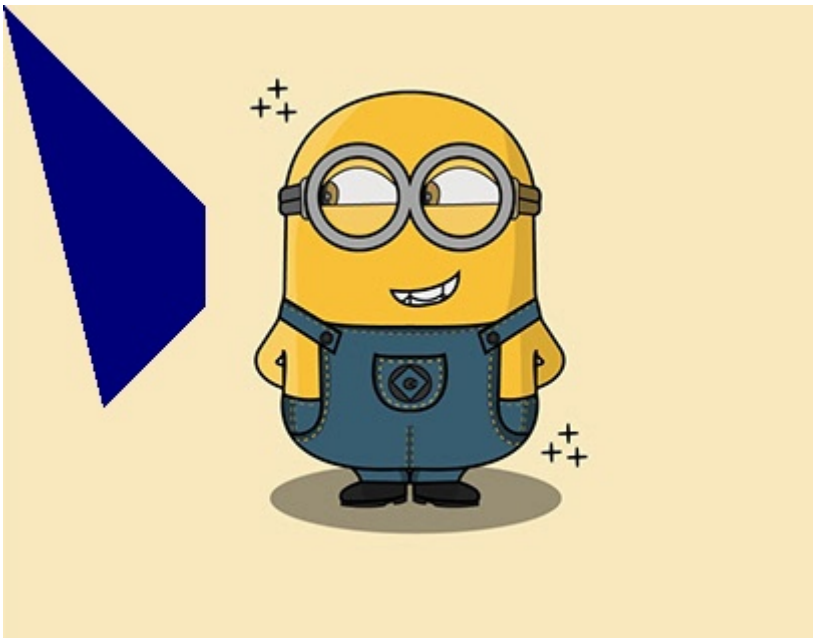
```
img = cv2.imread('img.jpeg')
pts = np.array([[0, 0], [100, 100], [100, 150], [50,200]] , np.int32)
isClosed = False
color = (120,0,0)
thickness = 3
cv2.polylines(img,[pts],isClosed,color,thickness)

cv2.imwrite("saved.jpeg",img)
```



- Для рисования залитого многоугольника необходимо использовать метод fillPoly. Он принимает на вход изображение - image; набор точек, которые будут последовательно соединены - [pts]; цвет - color. Данный метод изменяет переданное изображение.

Пример кода для рисования залитого многоугольника:

```
img = cv2.imread('img.jpeg')
pts = np.array([[0, 0], [100, 100], [100, 150], [50,200]] , np.int32)
color = (120,0,0)
cv2.fillPoly(img,[pts],color)
cv2.imwrite("saved.jpeg",img)
```



Библиотека OpenCV -> Работа с цветом

Для получения пикселей изображения необходимо использовать оператор [] для изображения (numpy-массива).

Например, чтобы получить пиксель 3-й строки, 5-го столбца можно написать:

pixel = img[3,5]

Для изменения пикселя на другой необходимо изменить элемент numpy-массива, который представляет из себя изображение:

img[3,5] = [255,0,0].

Пиксель представляет из себя numpy-массив из 3-х элементов (при изменении пикселя можно передавать list).

OpenCV не умеет работать с альфа-каналом изображений и с палитрой цветов.

При считывании изображения, где используется палитра цветов, OpenCV, при создании numpy массива, сразу записывает вместо индексов цветов палитры сами цвета. При считывании изображения с альфа-каналом - он отбрасывается.

Библиотека OpenCV -> Работа с областями изображений, работа с текстом №1

Работа с областями изображения. OpenCV.

Помимо библиотеки OpenCV есть вспомогательный пакет imutils, который основан на OpenCV. Он позволяет упростить некоторые операции с изображениями.

Установить его можно с помощью утилиты pip:

```
pip install imutils
```

Подключить его к кода можно следующим образом:

```
import imutils
```

Рассмотрим способы взаимодействия с областями изображений с помощью библиотеки OpenCV и imutils:

- cv2.resize - данный метод позволяет изменить размер изображения. Первым аргументом принимает изображение, втором – размер итогового изображения. Также необязательным параметром является метод интерполяции, с помощью него определяется каким способом будут заполняться пиксели нового изображения.

Пример кода уменьшения высоты и ширины изображения в 1.5 раза:

```
img = cv2.imread("img.jpeg")
height = img.shape[0]
width = img.shape[1]
small_img = cv2.resize(img, (int(width/1.5), int(height/1.5)))
```

Метод cv2.resize возвращает новое изображение, а не изменяет переданное.

- Также метод resize есть в пакете imutils. Этот метод также позволяет изменить размер изображения, но работает он по другому. При изменении изображения часто необходимо сохранить пропорции изображения, функция resize в imutils позволяет сохранить соотношение сторон, она принимает изображение и либо высоту (height), либо ширину (width) до которой нужно изменить изображение. Если передать высоту, то ширина автоматически изменится под изначальные пропорции, как и наоборот, если передать ширину - высота изменится под изначальные пропорции.

Сравнение resize из imutils и из OpenCV:

```
img = cv2.imread("img.jpeg")
print("Изначальные размеры:",img.shape[0:2])

small_img_cv2 = cv2.resize(img,(100,100))
print("Размеры после применения cv2.resize:",small_img_cv2.shape[0:2])

small_img_imutils = imutils.resize(img,width=100)
print("Размеры после применения imutils.resize:",small_img_imutils.shape[0:2])
```

```
Изначальные размеры: (316, 404)
Размеры после применения cv2.resize: (100, 100)
Размеры после применения imutils.resize: (78, 100)
```

- Для обрезания изображения, можно использовать, например, такую функцию:

```
def crop(img,x1,y1,x2,y2):
    return img[y1:y2,x1:x2]
```

x1,y1 – координата верхнего левого угла области, которую нужно обрезать (включительно);

x2,y2 – координаты нижнего правого угла области (не включительно).

- Для поворота изображения или его части можно воспользоваться методом cv2.rotate(). Данный метод позволяет выполнить поворот на 90,180 или 270 градусов. Методу нужно передать изображение и одну из констант, которая отвечает за то, насколько градусов нужно совершить поворот.

Данный метод возвращает новое изображения, не изменяет исходное.

Константы:

1. cv2.ROTATE_90_CLOCKWISE - поворот на 90 градусов по часовой стрелке
2. cv2.ROTATE_180 - поворот на 180 градусов
3. cv2.ROTATE_90_COUNTERCLOCKWISE - поворот на 270 градусов по часовой стрелке

Пример кода для поворота всего изображения на 180 градусов:

```
img  = cv2.imread("img.jpeg")
img  = cv2.rotate(img,cv2.ROTATE_180)
```

Пример поворота части изображения на 90 градусов по часовой стрелке:

```
img = cv2.imread("img.jpeg")
img[100:200,100:200] = cv2.rotate(img[100:200,100:200],cv2.ROTATE_90_CLOCKWISE)
cv2.imwrite("rotated.jpeg",img)
```

- Для поворота изображения на любое количество градусов можно использовать метод rotate из imutils. Первым аргументом ему необходимо передать изображение, вторым - angle – угол поворота. Поворот изображения производится против часовой стрелки.

Данный метод не изменяет исходное изображение, а возвращает новое.

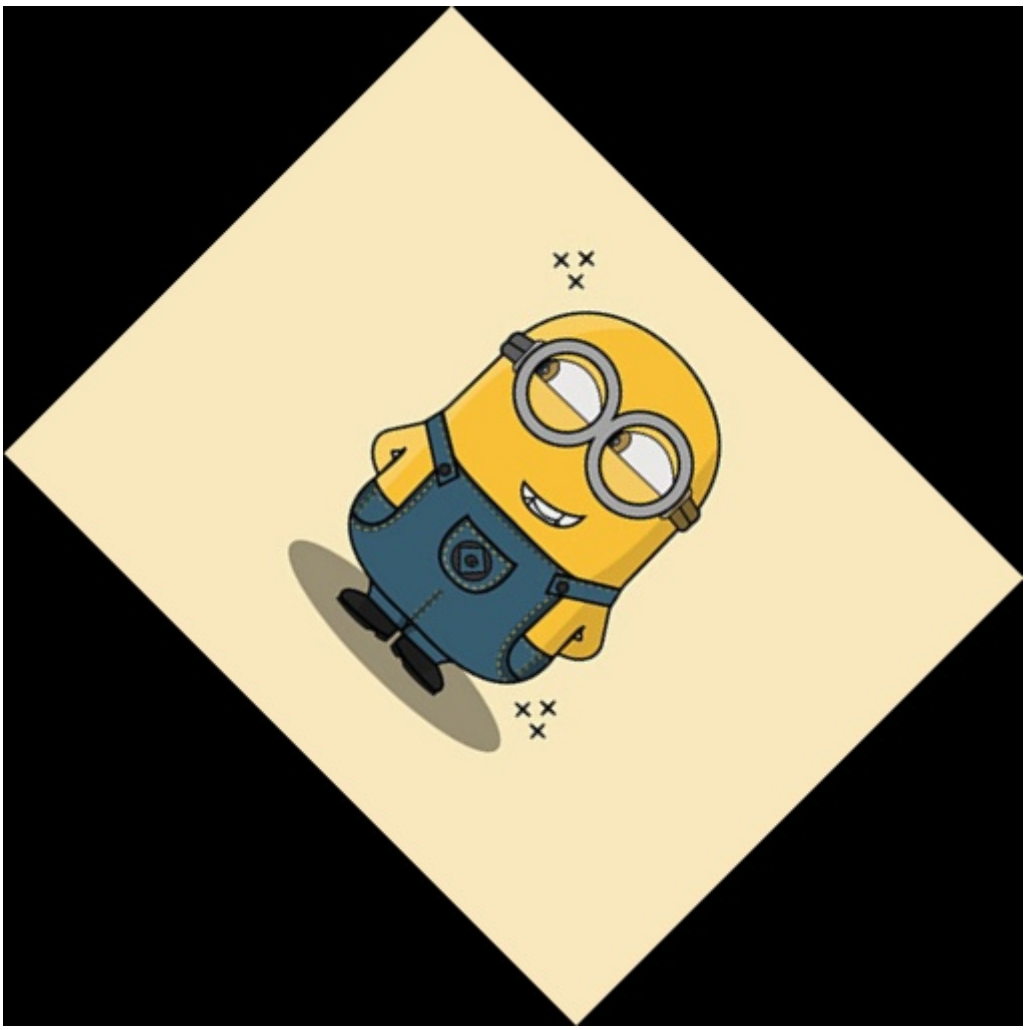
Пример поворота на 45 градусов:

Можно увидеть, что углы изображения обрезались, так как размер изображения остается прежним.

Для того, чтобы размеры изображения изменились так, чтобы всё исходное изображение поместилось можно использовать другой метод – rotate_bound. Он принимает такие же аргументы, но поворачивает уже по часовой стрелке.

Пример:

```
img = cv2.imread('img.jpeg')
rotated_img = imutils.rotate_bound(img,angle = 45)
cv2.imwrite('rotated_img.jpeg',rotated_img)
```



- Отзеркалить изображение или его часть можно с помощь метода cv2.flip.

В качестве первого аргумента передается изображение. Также нужно передать аргумент flipCode – целое число.

Если flipCode>0, то отзеркаливание происходит по вертикальной оси.

Если flipCode равен 0, то отзеркаливание происходит по горизонтальной оси.

Если flipCode<0, то по обоим осям сразу.

Данный метод не изменяет текущее изображение, а возвращает новое.

Пример кода, для отзеркаливания изображения по вертикальной оси:

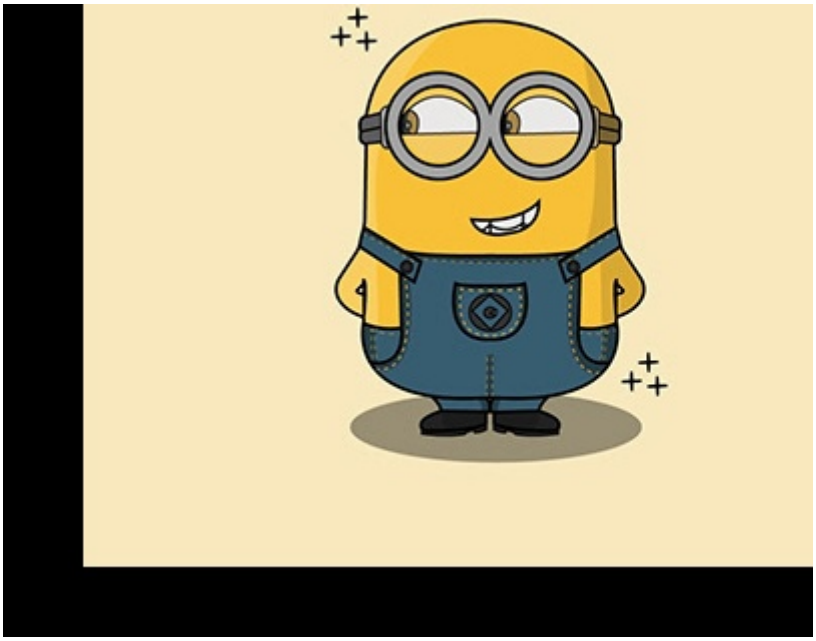
```
img = cv2.imread("img.jpeg")
flipped_img = cv2.flip(img,flipCode=1)
cv2.imwrite('flipped_img.jpg', flipped_img)
```

- Для вставки одного изображения в другое, или части изображения в другое изображение нужно использовать оператор [], аналогично обрезанию изображения (выше).
- Произвести сдвиг изображения можно с помощью метода translate из imutils. В качестве первого аргумента ему передается изображение, затем сдвиг по горизонтальной оси и сдвиг по вертикальной оси.

Данный метод возвращает новое изображение, не изменяет исходное.

Пример использования:

```
img = cv2.imread("img5.jpeg")
translated = imutils.translate(img, 40, -35)
cv2.imwrite("translated.jpeg",translated)
```



Библиотека OpenCV -> Работа с областями изображений, работа с текстом №2

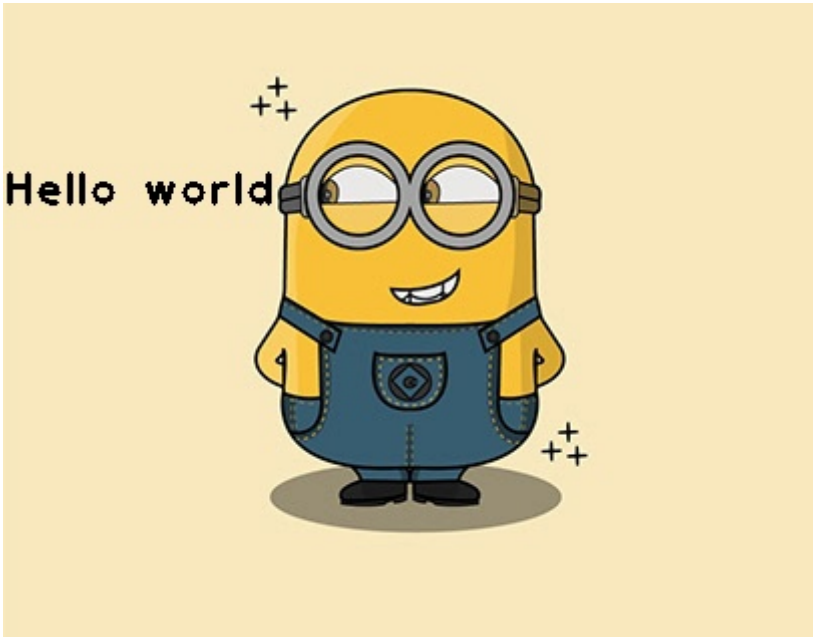
Работа с текстом. OpenCV.

Также с помощью библиотеки OpenCV можно вставлять текст в картинку.

Для добавления текста на изображение используется метод putText. Он принимает на вход изображение – image; текст – text; org – нижняя левая координата текста; шрифт – font; масштабирование шрифта – fontScale; цвет – color; толщина текста – thickness;

Пример использования:

```
img = cv2.imread("img5.jpeg")
text = 'Hello world'
org = (0,100)
font = cv2.FONT_HERSHEY_PLAIN
fontScale = 1.5
color = (0,0,0)
thickness = 2
cv2.putText(img,text,org,font,fontScale,color,thickness)
cv2.imwrite("img_with_text.jpeg",img)
```



Работа с файлами и заголовками изображений -> Работа с файлами №1

Работа с файлами

Документация по работе с файлами:
https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

Работа с файлами - это один из важных аспектов программирования. Файлы используются для хранения и обмена данными между программами, а также для сохранения результатов работы программы.

В Python для работы с файлами используются встроенные функции, такие как: open(), read(), write(), close().

Функция open() открывает файл и возвращает объект файла (io.TextIOWrapper), который может использоваться для чтения или записи данных.

Пример открытия файла на чтение:

```
file = open('path.txt','r')
```

В данном примере открывается файл "path.txt" на чтение. Объект файла записывается в переменную file. Второй аргумент 'r' указывает, что файл открывается для чтения.

Вместо 'r' могут быть следующие варианты:

- 'r' - Чтения из файла (устанавливается по умолчанию)
- 'w' - Запись в файл. Если файл не существует, то он будет создан. Стирает все данные файла, которые были в нем до открытия.
- 'x' - Запись в файл. Если не существует, то будет выброшено исключение. Стирает все данные файла, которые были в нем до открытия.
- 'a' - Запись в файл. Не стирает данные, которые были до открытия, а добавляет новые в конец файла.
- 't' - Открывает как текстовый файл (устанавливается по умолчанию)
- 'b' - Открывает как двоичный файл
- "+" - Позволяет работать с файлом как в режиме чтения, так и в режиме записи

Режимы можно совмещать. Например, 'wt' - откроет файл как текстовый на запись. 'rb' - откроет файл как бинарный на чтение.

У объекта файла есть атрибуты, с помощью которых можно получать информацию о файле:

- name - название файла
- mode - режим, в котором этот файл открыт
- closed - возвращает True, если файл был закрыт

После выполнения всех необходимых действий с файлом его нужно закрыть. Делается это с помощью метода close() объекта файла.

```
file.close()
```

Когда файл открывается с помощью метода open(), операционная система выделяет определенное количество ресурсов для работы с файлом, таких как оперативная память и процессорное время. Если не закрыть файл методом close(), то эти ресурсы будут заняты до тех пор, пока программа не завершится или не закроет файл. Это может привести к нехватке ресурсов и замедлению работы программы.

Кроме того, если файл не закрыт, то для других программ может быть ограничен доступ к этому файлу.

При закрытии файла методом close() освобождаются все ресурсы системы, которые были выделены для операции с файлом.

Также безопасно работать с файлами позволяет конструкция with...open(). Она позволяет закрыть файл после завершения работы с ним, даже если в процессе работы программы произошла ошибка.

Пример использования:

```
with open('file.txt','r') as file:
    data = file.read()
```

В данном примере открывается файл "file.txt" для чтения с помощью конструкции with...open(). Файл будет автоматически закрыт после выполнения блока кода внутри конструкции. Данная конструкция является более безопасной и удобной альтернативой использованию метода close().

Чтение файла

Для чтения файла можно использовать функцию read() объекта файла.

Пример:

```
with open('file.txt', 'r') as file:
    text = file.read()
```

В данном случае будет прочитан весь текст файла целиком и все данные в нем запишутся в одну строку - text.

Также с помощью функции read() можно считывать данные файла порционно:

```
with open('file.txt', 'r') as file:
    text = file.read(20)
```

В этом случае в переменную text запишутся первые 20 символов текста, записанные в файле.

При применении этой функции несколько раз подряд будет считываться порция за порцией этого текста. Так как виртуальный курсор каждый раз будет сдвигаться по файлу. Этот курсор также можно подвинуть на нужное место, без считывания - с помощью функции seek():

```
with open('file.txt', 'r') as file: # "It's txt file"
    text_1 = file.read(6) # "It's t"
    file.seek(3)
    text_2 = file.read(9) # "s txt fil"
```

Также файл можно считывать построчно, с помощью метода readline() объекта файла. Данный метод считывает файл аналогично методу read(), но построчно, а не посимвольно.

Метод readlines() возвращает все строки файла в виде списка (list).

Также считывать строки с файла можно с помощью итератора объекта файла, пример:

```
with open('file.txt','r') as file:
    for line in file:
        print(line)
```

В данном примере считываются и выводятся на экран все строки файла.

Запись в файл

Для записи в файл его необходимо открыть в режиме "w", "a", "+" или "x".

Записать что-либо в файл можно с помощью метода write() объекта файла:

```
with open('file.txt', 'w') as file:
    file.write('Hello world!\nHello Python!')
```

Данный код перезаписывает данные в файле "file.txt", записывая туда 2 строки - "Hello world!" и "Hello Python!".

В качестве аргумента метода write() может быть только строка. Если необходимо записать туда другой объект, его необходимо привести к строковому типу.

Также в файл можно записать множество строк из списка, с помощью функции writelines() объекта файла.

```
with open('file.txt', 'w') as file:
    file.writelines(['Hello world!\n', 'Hello Python!'])
```

Еще одним способом записи в файл является функция print(). С помощью нее можно записать в файл любую информацию, причем в функцию print() необязательно передавать только строки - можно передавать любые типы данных, они автоматически преобразуются в строковый тип. Для того, чтобы с помощью функции print записать данные в файл, а не в консоль, ей необходимо передать аргумент file:

```
with open('file.txt', 'w') as file:
    print("Hello world!", file=file)
```

Работа с файлами и заголовками изображений -> Работа с файлами №2

Модуль os

Стандартный модуль os предоставляет функции для работы с операционной системой, включая файловую систему. Данный модуль содержит множество функций для работы с директориями и файлами.

Рассмотрим некоторые функции, которые предоставлены данным модулем:

- Создание директории:

Данный код создаст новую директорию (папку) с названием 'new_directory'

```
import os
os.mkdir('new_directory')
```

- Удаление директории

Данный код удалит директорию с названием 'new_directory'

```
import os
os.rmdir('new_directory')
```

При попытке удалить несуществующую директорию или создать директорию с уже существующим названием выбросится исключение.

- Удалить файл можно с помощью метода os.remove(path) - передав ему путь к файлу
- Перемещение файла или директории.

Для переименования или перемещения файла или директории может быть использована функция os.rename(). Первый аргументом функция принимает путь к файлу/директории, которую нужно переместить, вторыми аргументом - новый путь:

```
import os
os.rename('new_directory','directory/new_directory')
```

Данный код переместит директорию 'new_directory' в папку 'directory'.

Работа с путями файловой системы

Работа с путями файловой системы осуществляется с помощью модуля os.path.

В различных операционных системах могут использоваться различные разделители. Например в Windows используется обратный слеш "\", а в Linux или MacOS - прямой слеш "/".

Функция os.path.join() автоматически использует правильные разделители при построении пути. Это позволяет избежать проблем с разделителями и делает код более переносимым между различными операционными системами.

Так, например, чтобы получить путь из текущей директории в какую либо ниже, можно воспользоваться следующим кодом:

```
os.path.join(".", "some_directory", "other_directory")
```

Данный код, для операционной системы Windows вернет следующую строку : ".\some_directory\other_directory"

Строка "." означает доступ к текущей директории. Строка ".." - доступ к директории выше.

Так, чтобы получить путь в директорию двумя уровнями выше, можно использовать следующий код:

```
os.path.join(".","..", "..")
```

Чтобы вывести все папки и файлы директории в виде списка, можно воспользоваться функцией os.listdir(), которая принимает в качестве аргумента путь к директории. Если вызвать функцию без аргументов, она вернет список файлов и папок для текущей директории.

Для проверки существует ли такая папка или файл можно воспользоваться функцией os.path.exists(), она принимает на вход путь к папке или директории и возвращает True, если файл/директория существует, иначе - False.

Функция os.walk() позволяет выполнить рекурсивный обход всего содержимого папки. Пример обхода текущей директории:

```
for current_dir, dirs, files in os.walk("."):
    print(current_dir, dirs, files)
```

current_dir - путь к текущей папке (строка)

dirs - список всех папок в текущей папке (список строк)

files - список всех файлов в текущей папке (список строк)

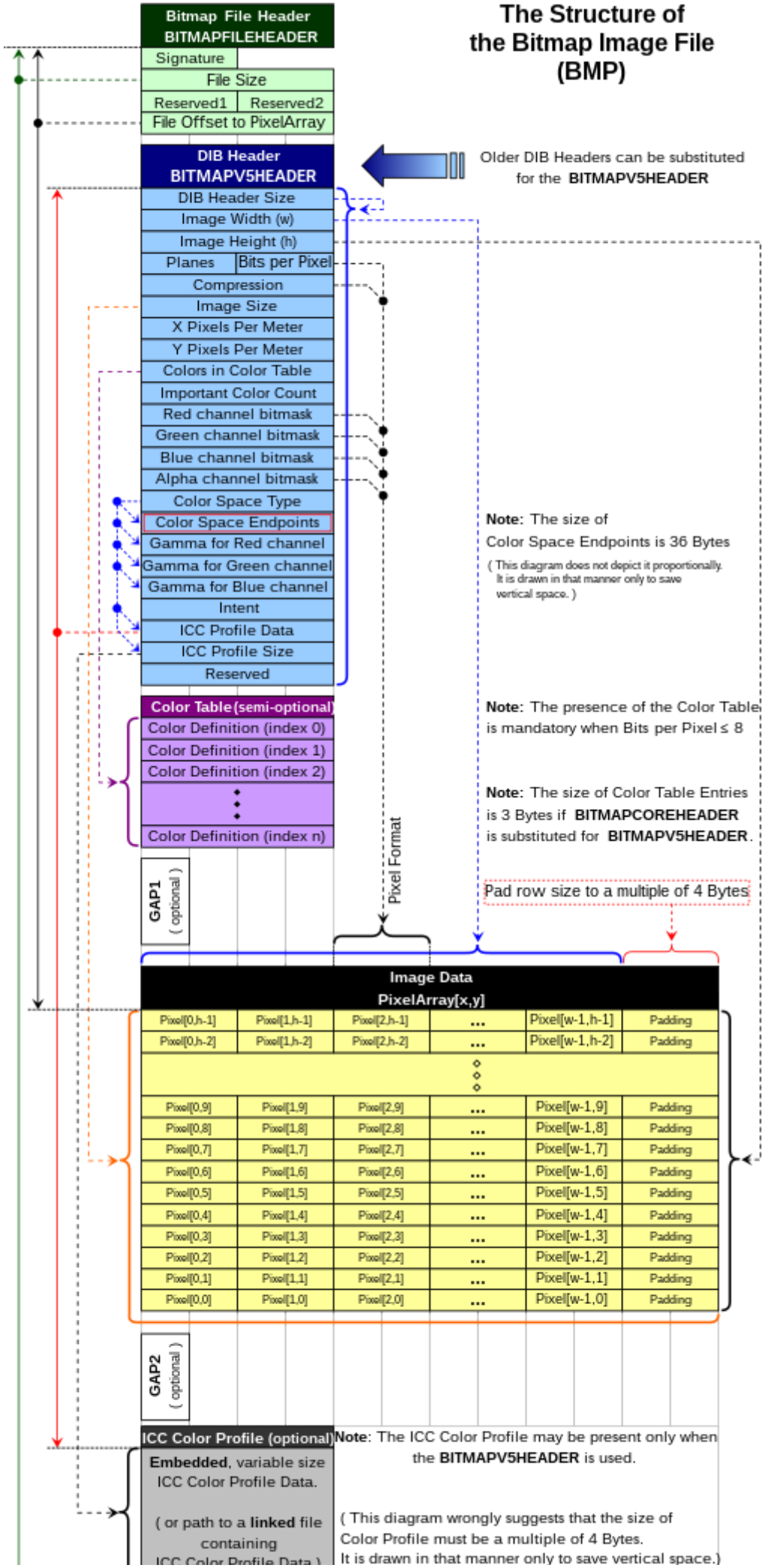
Работа с файлами и заголовками изображений -> Работа с заголовками изображений №1

Формат BMP

BMP файлы – формат хранения растровых изображений. Файлы с этим форматом имеют расширение ".bmp".

Пиксели изображения данного формата могут состоять из различного количества бит. Частичная прозрачность изображения реализована альфа-каналом битностей от 16 и выше.

Рассмотрим структуру файла формата BMP:


The Structure of the Bitmap Image File (BMP)
Bitmap File Header
BITMAPFILEHEADER
Signature
File Size
Reserved1
Reserved2
File Offset to PixelArray
DIB Header
BITMAPV5HEADER
Older DIB Headers can be substituted for the BITMAPV5HEADER
DIB Header Size
Image Width (w)
Image Height (h)
Planes
Bits per Pixel
Compression
Image Size
X Pixels Per Meter
Y Pixels Per Meter
Colors in Color Table
Important Color Count
Red channel bitmask
Green channel bitmask
Blue channel bitmask
Alpha channel bitmask
Color Space Type
Color Space Endpoints
Gamma for Red channel
Gamma for Green channel
Gamma for Blue channel
Intent
ICC Profile Data
ICC Profile Size
Reserved
Note: The size of Color Space Endpoints is 36 Bytes
(This diagram does not depict it proportionally. It is drawn in that manner only to save vertical space.)
Color Table (semi-optional)
Color Definition (index 0)
Color Definition (index 1)
Color Definition (index 2)
Color Definition (index n)
Note: The presence of the Color Table is mandatory when Bits per Pixel ≤ 8
Note: The size of Color Table Entries is 3 Bytes if BITMAPCOREHEADER is substituted for BITMAPV5HEADER.
GAP1 (optional)
Pixel Format
Pad row size to a multiple of 4 Bytes
Image Data
PixelArray[x,y]
Pixel[0,h-1] Pixel[1,h-1] Pixel[2,h-1] ... Pixel[w-1,h-1] Padding
Pixel[0,h-2] Pixel[1,h-2] Pixel[2,h-2] ... Pixel[w-1,h-2] Padding
Pixel[0,9] Pixel[1,9] Pixel[2,9] ... Pixel[w-1,9] Padding
Pixel[0,8] Pixel[1,8] Pixel[2,8] ... Pixel[w-1,8] Padding
Pixel[0,7] Pixel[1,7] Pixel[2,7] ... Pixel[w-1,7] Padding
Pixel[0,6] Pixel[1,6] Pixel[2,6] ... Pixel[w-1,6] Padding
Pixel[0,5] Pixel[1,5] Pixel[2,5] ... Pixel[w-1,5] Padding
Pixel[0,4] Pixel[1,4] Pixel[2,4] ... Pixel[w-1,4] Padding
Pixel[0,3] Pixel[1,3] Pixel[2,3] ... Pixel[w-1,3] Padding
Pixel[0,2] Pixel[1,2] Pixel[2,2] ... Pixel[w-1,2] Padding
Pixel[0,1] Pixel[1,1] Pixel[2,1] ... Pixel[w-1,1] Padding
Pixel[0,0] Pixel[1,0] Pixel[2,0] ... Pixel[w-1,0] Padding
GAP2 (optional)
ICC Color Profile (optional)
Embedded, variable size ICC Color Profile Data.
(or path to a linked file containing ICC Color Profile Data)
Note: The ICC Color Profile may be present only when the BITMAPV5HEADER is used.
(This diagram wrongly suggests that the size of Color Profile must be a multiple of 4 Bytes. It is drawn in that manner only to save vertical space.)

Файл BMP по сути представляет собой набор байт, идущих друг за другом. В заголовке данные сгруппированы в группы по 2 или 4 байта.

Для того, чтобы среда, например, операционная система, умела работать с файлами формата BMP она должна уметь читать и интерпретировать данную структуру файла.

Рассмотрим заголовок файла:

- Первые 2 байта – Signature. В BMP файле здесь хранятся байты (42 4D), они интерпретируются как символы "BM" (таблица ASCII). С помощью них среда поймет, что работает с форматом BMP.
- Затем идут 4 байта – File Size. Это размер файла, с помощью него среда может понять, какую область памяти необходимо читать.
- Reserved1, Reserved2 – по 2 байта, зарезервированные поля, для возможного добавления функционала, изменения версий.
- File Offset to PixelArray – 4 байта. Количество байт до начала массива пикселов.

Затем расположен DIB header. Рассмотрим его важные характеристики:

- DIB Header Size – 4 байта, размер DIB заголовка.
- Image Width (w), Image Height (h) – ширина и высота изображения в пикселях, по 4 байта.
- Bit per Pixel – 2 байта, количество бит на один пиксель.
- Image Size – размер массива пикселов в байтах.
- Colors in Color Table – количество цветов в палитре цветов .

В памяти строки пикселей хранятся в обратном порядке – сначала нижняя, в конце верхняя.

В каждой строке заголовка байты также записаны в обратном порядке. Так, например, если Image Width = “00 02 00 00”, то для получения ширины в десятичном формате, необходимо перевернуть данную строку байт и получить десятичное значение. “00 02 00 00” -> “00 00 02 00” -> 200_{16} = 512_{10} пикселей.

Чтобы добраться до ширины (Image Width) изображения в пикселях, необходимо начать читать файл сначала, пропустить первые 18 байт (Signature (2 байта), File Size (4 байта), Reserved1(2 байта), Reserved2(2 байта), File Offset to PixelArray(4 байта), DIB Header Size (4 байта)). Затем считать 4 байта.

После DIB Header расположена палитра (таблица) цветов.

О палитре цветов:

Если цветов не много, то в файле может использоваться таблица (палитра) цветов. Обычно цвет кодируется 24 битами, но если количество цветов в изображении, например, всего 4. То для хранения каждого пикселя можно использовать всего 4 бита.

В этом случае каждому из 4 цветов назначается 2-битный индекс, а соответствие индекс-цвет хранится в таблице цветов. Вместо 24-битной записи в массиве пикселей используются индексы.

Работа как с бинарным файлом с помощью Python

Файлы изображений различных форматов можно считывать в виде бинарных файлов с помощью стандартной функции языка Python open().

Пример считывания файла в бинарном виде:

```
img_bin = open('img.bmp','rb')
```

В качестве первого аргумента передается путь к файлу и его имя, в качестве второго – режим открытия файла. Символ ‘r’ означает открыть для чтения (если ‘w’ то для записи), ‘b’ – бинарный формат.

Рассмотрим bmp-файл в бинарном виде:

```
with open('peppers.bmp','rb') as fbmp:
    for i in fbmp:
        print(i.hex(' '))
```

```
42 4d 36 00 0c 00 00 00 00 00 36 00 00 00 28 00 00 00 02 00 00 00 02 00 00 01 00 18 00 00 00 00 00 00 00 00 00 00 12 0b 00 00
12 0b 00 00 00 00 00 00 00 00 00 00 00 00 04 00 01 01 00 04 00 00 06 00 00 08 00 00 09 00 00 0b 00 00 0d 00 00 0d 00 00 0f 00
00 0f 00 00 0f 00 00 0e 01 00 0d 06 00 0c 0c 00 0b 12 00 0a
16 00 09 1a 00 09 1b 00 08 1d 00 08 1f 00 08 1f 00 08 20 00 08 21 00 08 21 00 07 23 00 07 23 00 07 25 00 06 27 00 06 28 00 06
2a 00 05 2c 00 05 2e 00 05 2e 00 04 30 00 04 30 00 03 30 00 03 31 00 03 30 00 02 31 00 02 31 00 01 32 00 01 32 00 01 32 00 01
33 00 01 33 00 01 33 00 00 34 00 01 32 00 01 32 00 01 32 00 02 31 00 01 32 00 01 32 00 01 33 00 01 33 00 00 34 00 00 34 00 00
36 00 00 36 00 00 37 00 00 37 00 00 37 00 00 37 00 00 37 00 01 35 00 00 35 00 01 34 00 01 34 00 01 34 00 01 35 00 01 36 00 01
36 00 01 35 00 01 35 00 01 34 00 01 35 00 00 36 00 00 37 00 00 38 00 00 39 00 00 3c 00 00 3d 00 00 3e 00 00 3d 00 00 3d 00 00
3c 00 00 3c 00 00 3c 00 00 3b 00 00 3c 00 00 3c 00 00 3d 00 00 3d 00 00 3c 00 00 3b 00 00 3a 00 00 37 00 00 35 00 00 30 00 00
2b 00 00 26 00 02 1e 00 04 17 00 07 0e 00 0a
05 00 0d 00 00 0f 00 00 10 00 00 11 00 00 10 00 00 0f 00 00 10 00 00 12 00 00 13 00 00 15 00 00 15 00 00 16 00 00 16 00 00 15
00 00 13 00 00 13 00 00 13 00 00 12 00 00 11 00 00 0e 02 00 0b 0d 00 08 18 00 04 23 00 03 2b 00 00 30 00 00 31 00 00 33 00 00
33 00 00 35 00 00 36 00 00 37 00 00 37 00 00 36 00 00 36 00 00 36 00 00 36 00 00 36 00 00 38 00 00 38 00 00 39 00 00 38 00 00
38 00 00 39 00 00 3a 00 00 3a 00 00 3a 00 00 3a 00 00 3a 00 00 38 00 00 37 00 00 30 00 00 2a 00 02 21 00 03 1b 00 05 15 00 05
14 00 05 13 00 06 12 00 06 0f 00 08 0a
00 0a
```

Первыми, как было сказано выше, идут байты 42 и 4d.

Получим размер файла в байтах - он хранится в байтах со 2-го по 5-й включительно (начиная с 0). Байты каждый строки, кроме Signature, хранятся в обратном порядке. Так, например, размер файла записан как "36 00 0c 00". Чтобы получить размер в байтах нужно перевернуть побайтово строку, перевести ее в десятичное число. Получим «36 00 0C 00» - > $C0036_{16}$ -> 786486_{10} байт.

Сделаем то же самое с помощью кода Python:

```
with open('peppers.bmp','rb') as fbmp:
    from_, to_ = 2,6
    img_size_arr = fbmp.read().hex(' ').split(' ')[from_ : to_]
    img_size_arr.reverse()
    print('img_size_arr: ',img_size_arr)

    img_size_str = ''.join(img_size_arr)
    print('img_size_str:',img_size_str)

    img_size_int = int(img_size_str,16)
    print('img_size_int:',img_size_int)
```

```
img_size_arr:  ['00', '0c', '00', '36']
img_size_str: 000c0036
img_size_int: 786486
```

Работа с файлами и заголовками изображений -> Работа с заголовками изображений №2

Формат PNG

PNG файлы – формат хранения растровых изображений. Файлы с этим форматом имеют расширение .png.

Рассмотрим структуру PNG – файла.

Файл содержит подпись “PNG signature” и критические блоки (чанки). Каждый из чанков несет в себе некоторую информацию о файле.

PNG signature - это подпись файла, она всегда одинакова, состоит из 8 байт и представляет собой в hex-записи – (89 50 4E 47 0D 0A 1A 0A).

- 89 – non-ASCII символ. Препятствует распознаванию PNG как текстового файла.
- 50 4E 47 – “PNG” в ASCII записи. Чтобы среда понимала, что работает с файлом формата PNG.
- 0D 0A - DOS-style перевода строки
- 1A – останавливает вывод файла в DOS режиме (end of line)
- 0A – Unix-style перевод строки.

Chunks.

Чанки – это блоки данных из которых состоит файл. Каждый чанк состоит из 4-х секций:

Length (длина)	Тип (имя) чанка	Содержание чанка	CRC
4 байта	4 байта	*Length* байт	4 байта

1. Length (4 байта) – длина чанка в байтах
2. Тип чанка (4 байта) – 4 ASCII символа. Каждый символ – это флаг, который сообщает декодеру некоторую дополнительную информацию.

Регистр первого символа говорит о том критический это чанк (верхний регистр) или вспомогательный (нижний регистр). Критические чанки должны распознаваться каждым декодером изображения и если он не может его распознать, он должен закончить выполнение с ошибкой.

Регистр второго символа задает публичность (верхний) или приватность (нижний).

Третий символ зарезервирован (оставлен для будущих свершений)

Регистр четвертого символа означает возможность копирования данного чанка редакторами, которые не могут его распознать. Если регистр нижний, чанк может быть скопирован, иначе он копируется только в случае, когда при модификации не были затронуты критические чанки.

Пример:

1. Данные чанка
2. Контрольная сумма

Основные критические чанки:

- IHDR – заголовок файла, содержит основную информацию о изображении. Обязан быть первым чанком.
- PLTE – палитра цветов
- IDAT – содержит изображение. Изображение можно быть разбито на несколько IDAT чанков для потоковой передачи.
- IEND – завершающий чанк, должен быть последним в файле.

Пример вспомогательных чанков:

- bkGD – задает основной фоновый цвет
- hIST – в этом чанке может находить гистограмма цветов
- tIME – хранит дату последнего изменения изображения

Минимальный обязательный состав PNG файла представлен на рисунке:

Рассмотрим содержания чанка IHDR:

- ширина изображения – 4 байта
- высота – 4 байта
- битовая глубина – 1 байт.
- тип цвета – 1 байт, состоит из 3-х флагов. 1(используется палитра), 2(используется цвет, не монохромное изображение), 4 (присутствует альфа-канал).
- метод сжатия – 1 байт
- метод фильтации – 1 байт
- interlace – порядок передачи данных

Данные в IDAT хранятся в закодированном (сжатом) виде.

Работа как с бинарным файлом с помощью Python

Рассмотрим, как выглядит PNG файл в бинарном виде:

```
with open('lion.png','rb') as img:
    for line in img:
        print(line.hex(' '))
```

```
89 50 4e 47 0d 0a
1a 0a
00 00 00 0d 49 48 44 52 00 00 00 b7 00 00 01 13 08 03 00 00 00 43 4a 67 e1 00 00 00 81 50 4c 54 45 00 00 00 ff ff ff eb eb eb
f4 f4 f4 fc fc fc f7 f7 f7 cc cc cc d0 d0 d0 c1 c1 c1 dd dd dd a9 a9 a9 e8 e8 e8 d7 d7 d7 c3 c3 c3 6f 6f 6f 8f 8f 8f 4f 4f 4f
a2 a2 a2 e1 e1 e1 55 55 55 b8 b8 b8 3e 3e 3e 82 82 82 a5 a5 a5 95 95 95 49 49 49 9b 9b 9b 43 43 43 b2 b2 b2 5a 5a 5a 65 65 65
27 27 27 87 87 87 30 30 30 7b 7b 7b 29 29 29 32 32 32 1a 1a 1a 75 75 75 15 15 15 68 68 68 0c 0c 0c 18 18 18 c9 ce 20 99 00 00
11 2d 49 44 41 54 78 9c ed 5d e7 7a ea 30 12 35 60 6a e8 2d 40 68 21 21 65 df ff 01 17 49 33 a3 19 49 36 e4 62 23 f6 5b 9f 1f
f7 c6 fd 58 1e 4d b7 49 92 0a
15 2a 54 a8 50 a1 42 85 0a
15 2a 54 a8 50 a1 42 85 0a
15 2a 54 a8 50 a1 42 85 0a
15 2a 3c 06 87 b7 d9 fa 23 36 89 3f 62 3d e9 d6 34 d2 c1 38 36 97 9b 71 e8 35 6a 16 cd 43 6c 3e b7 e1 30 a8 71 6c 62 f3 b9 11
43 c1 ba fe 1e 9b cf 75 fc 9e be 93 64 2f 68 77 62 73 ba 8a d3 a0 59 6b 5f fe 17 b4 57 de 6e ef e3 dd 7e f6 f6 78 7a 19 d8 37
15 cd 45 92 6c 39 ed 2f b1 cf 7a 35 a8 e3 96 b4 bd fa 8c 44 95 63 a9 c9 74 13 39 dc 7b b6 c7 be 93 d6 1c 4c bf b2 4e f7 28 2c
```

В первой и второй строке можно увидеть подпись, о которой говорилось ранее – «89 50 4E 47 0D 0A 1A 0A».

Затем идут 4 байта – размер содержимого чанка в байтах. «00 00 00 0d» - 13 байт.

Следующие 4 байта – «49 48 44 52» - имя чанка (16-ричные ASCII коды). 49 – I, 48 – H, 44 – D, 52 – R. Получаем- IHDR.

Затем идут 13 байт содержимого чанка.

Например ширина изображения в пикселях находится в первых 4-х байтах – «00 00 00 b7» -> $b7_{16}$ -> 183_{10} пикселя. Высота изображения - в следующих 4-х байтах.

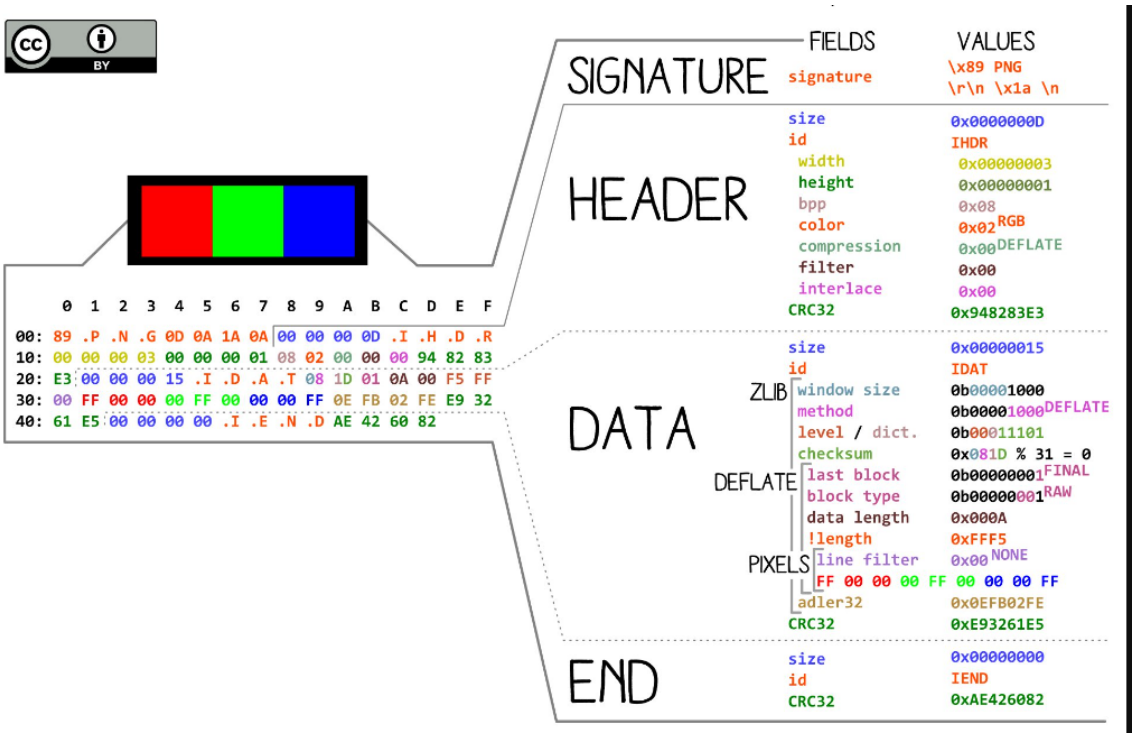
Рассмотрим пример считывания подписи Signature заголовка. Она находится в первых 8 байтах. Также можно заметить из рисунка выше, что она находится в первых 2-х строках файла.

Считаем ее:

```
with open('lion.png','rb') as img:
    signature = img.readline().hex(' ') + ' ' + img.readline().hex(' ')
    print('signature string: ',signature)

    png_arr = signature.split(' ')[1:4]
    png_arr_ascii = list(map(lambda x: bytes.fromhex(x).decode(),png_arr))

    png_str = ''.join(png_arr_ascii)
    print("ASCII: "+png_str)
```

```
signature string:  89 50 4e 47 0d 0a 1a 0a
ASCII: PNG
```

Изображение, иллюстрирующее расположение байт PNG-файла:



Работа с файлами и заголовками изображений -> Работа с заголовками изображений №3

Формат JPEG

JPEG файлы – формат хранения растровых изображений. Файлы с этим форматом имеют расширение .jpg (самое популярное), .jpeg, .jpe, .jfif.

Файл JPEG содержит последовательность маркеров, каждый из которых начинается с байта 0xFF, свидетельствующего о начале маркера, и байта-идентификатора. Некоторые маркеры состоят только из этой пары байтов, другие же содержат дополнительные данные, состоящие из двухбайтового поля с длиной информационной части маркера (включая длину этого поля, но за вычетом двух байтов начала маркера) и собственно данных. Такая структура файла позволяет быстро отыскать маркер с нужной информацией.

Данные пикселей изображения хранятся также, как и в формате PNG, в сжатом виде. Сжатие может производится как с потерями так и без.

Примеры маркеров:

- SOI. (Start Of Image) Первые 2 байта – FF D8. Больше не хранит данных, обозначает начало файла изображения.
- SOF0(Start Of Frame 0). Первые 2 байта – FF C0. Содержит информацию о размере изображения, количестве компонент цвета и других параметрах. Сегмент начинается с двух байтов FF C0, за которыми следует 2 байта, содержащие размер сегмента, 1 байт, содержащий точность (обычно 8), 2 байта, содержащие высоту изображения, 2 байта, содержащие ширину изображения, 1 байт, содержащий количество компонент цвета (обычно 3), и байты, содержащие информацию о каждом компоненте цвета. (Вместо SOF0 могут быть другие маркеры, например SOF1, с байтом-идентификатором C1 ; или SOF2, с байтом-идентификатором C2. Они используются, если изображение кодировалось не в базовом режиме).
- SOS (Start Of Scan). Первые 2 байта – FF DA. Являются разделяющим маркером между заголовком и закодированной (собственно сжатыми данными) информацией.
- COM. Первый 2 байта – FF FE. Комментарий. Содержит текст комментария.
- EOI (End Of Image) – FF D9 – конец закодированной части изображения.
- DHT (Define Huffman Table) - FFC4 - содержит таблицы Хаффмана, используемые при сжатии изображения.

- APP0 (Application Marker 0) - FFE0 - может содержать дополнительную информацию о файле, такую как производительность камеры, модель, дату и время съемки и так далее.

Работа как с бинарным файлом в Python

Рассмотрим как выглядит JPEG-файл в бинарном виде:

```
: with open('img5.jpeg','rb') as img:
    img_str = img.read().hex(' ')
    print(img_str)
```

```
ff d8 ff dd 00 04 00 00 ff ee 00 0e 41 64 6f 62 65 00 64 c0 00 00 00 01 ff c2 00 11 08 01 3c 01 94 03 00 11 00 01 11 01 02 11
02 ff db 00 c5 00 04 04 04 04 04 04 03 04 04 04 08 07 07 07 05 04 05 06 07 06 06 04 05 05 06 05 04 04 0c 0c 04 04 05 04 03 03
0c 0c 03 04 03 04 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0e 0e 0e 0e 0e 01 03 06 06 06 06 06 09 0d 0d 0e 09 0a 09
08 06 0e 0c 0e 0e 0e 0e 14 14 0e 0e 0d 14 14 14 14 14 14 0e 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
14 14 14 14 14 14 14 14 14 02 03 06 06 06 06 04 05 07 07 06 0e 0e 05 0e 0e 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 ff c4 01 15 00 01 00 02 03 01
01 01 00 00 00 00 00 00 00 00 00 00 05 06 01 04 07 03 02 08 01 01 00 02 03 01 01 00 00 00 00 00 00 00 00 00 00 00 03 04 01 02
05 06 07 10 00 02 02 02 00 05 03 03 03 04 03 00 00 00 00 00 02 04 01 03 00 05 11 12 13 14 50 10 15 31 06 20 35 21 32 40 22 23
30 33 34 70 80 11 00 02 01 01 04 03 0a 0a 05 06 09 09 09 00 00 00 01 02 03 11 00 04 12 21 13 22 31 05 23 32 33 41 42 51 61 71
91 10 14 50 52 62 72 81 92 a1 b1 20 24 43 82 d1 63 93 a2 a3 b2 c1 34 53 54 73 83 a4 d2 f0 f1 06 30 74 84 94 c2 c3 e1 e2 15 35
40 44 70 80 b3 d3 d4 12 01 00 01 03 03 02 05 04 03 01 01 01 00 00 00 00 01 11 00 21 31 41 51 61 10 71 20 50 81 91 f0 40 a1 b1
d1 30 c1 e1 f1 70 80 13 00 01 02 04 04 03 07 03 02 06 03 00 00 00 00 00 01 00 02 03 11 31 50 10 12 20 21 13 41 61 04 14 30 32
51 71 81 22 33 42 80 91 40 52 62 90 a1 b1 60 70 72 ff da 00 0c 03 00 00 01 10 02 10 00 00 01 ed 1e 7e f8 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03 24 7e fa 6f e9 b8 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 30 c4 34 9a 57 65 d6 f9 5a 60 0c 55 64 d7 4f 9f 3f 8d 49 b3 82 4c 7b 5c 86 62 e4 33
```

Первым маркером является маркер SOI с байтом-идентификатором D8.

Рассмотрим, как можно получить, например, количество компонент цвета в изображении из бинарного файла.

Эта информация хранится в маркере SOF0. Первыми 2-мя байтами данного маркера являются "FF C0". Затем, пропустив 7 байт, мы получим 1 байт - количество компонент цвета:

```
with open('img.jpeg','rb') as file:
    prev = 0
    temp = 0
    while(not(prev == b"\xFF" and temp == b"\xC0")):#ищем SOF0
        prev = temp
        temp = file.read(1)
    file.read(7)#пропускаем 7 лишних байт
    components = file.read(1)
    print('hex: ',components)
    print('dec: ',int.from_bytes(components,'big'))
```

```
hex:  b'\x03'
dec:  3
```

Или для изображения в оттенках серого:

```
with open('black_white.jpeg','rb') as file:
    prev = 0
    temp = 0
    while(not(prev == b"\xFF" and temp == b"\xC0")):#ищем SOF0
        prev = temp
        temp = file.read(1)
    file.read(7)#пропускаем 7 лишних байт
    components = file.read(1)
    print('hex: ',components)
    print('dec: ',int.from_bytes(components,'big'))
```

```
hex:  b'\x01'
dec:  1
```