Библиотеки. Часть 2

План урока

1. Обработка изображений
2. Растровые изображения
3. PIL. Установка библиотек
4. Модельный пример
5. Фильтры

Аннотация

В первом уроке, посвященном модулям в Python, мы упоминали PyPI — настоящий кладезь библиотек для задач из разных областей. Обработка изображений — одна из таких областей, причём очень обширная. С ней мы сегодня и познакомимся — у такого выбора есть целых три причины.

Обработка изображений

Вы не задумывались, почему работа с изображениями так важна и популярна? Давайте попробуем найти ответ на этот, вроде бы, простой вопрос.

Во-первых, люди, увлекающиеся фотографией — едва ли не самое многочисленное полупрофессиональное сообщество в мире. Его популярности очень способствует распространение смартфонов и сервисов по работе с фотографиями — таких как Instagram (https://www.instagram.com/) и Pinterest (https://ru.pinterest.com/).

Во-вторых, работа с видео сводится к работе с отдельными изображениями. Это относится и к профессиональным техникам наложения фильтров, и даже к работе с хромакеем (https://ru.wikipedia.org/wiki/%D0%A5%D1%80%D0%BE%D0%BC%D0%B0%D0%BA%D0%B5%D0%B9), без которой не обходится практически ни один современный фильм.

В-третьих, модель представления изображения в памяти компьютера довольно проста. Почти всегда это многомерный массив целых чисел, который легко представляется в виде списка списков. Даже на начальном этапе изучения программирования эта область интересна как для обучения, так и для применения на практике.

Пока мы оставим за кадром вопросы скорости обработки изображений. С ними можно поэкспериментировать самостоятельно — это позволит обсудить скорость выполнения компилируемого и интерпретируемого кода. Кстати, для замеров времени тоже есть модуль — `timeit`.

Растровые изображения

Мы будем работать с растровыми изображениями, представляющими собой массив (таблицу) пикселей разных цветов.

Давайте посмотрим вот на это изображение.

Если мы приблизим его, то увидим пиксели. Давайте увеличим глаз совы (кстати, её зовут Рианна).

Итак, изображение можно моделировать списком списков (двумерной таблицей, в которой лежат цвета). Осталось только подумать, как именно кодировать цвета.

Опыт работы со строками, где каждому символу соответствует свой код, должен подсказывать вам, что и с изображениями должно быть так же. Мы можем пронумеровать некоторое количество цветов и указывать их номера в нашем списке списков. Совокупность выбранных цветов будет называться **палитрой**.

В итоге, нам нужен способ преобразования цветов в целые числа. Мы воспользуемся одной из самых популярных моделей представления цвета — RGB (Red, Green, Blue). Каждый из цветов в этой модели представляется как совокупность трёх компонентов: красного, синего и зеленого. Значение каждого компонента лежит в диапазоне от 0 (минимум) до 255 (максимум), занимая таким образом 1 байт в памяти.

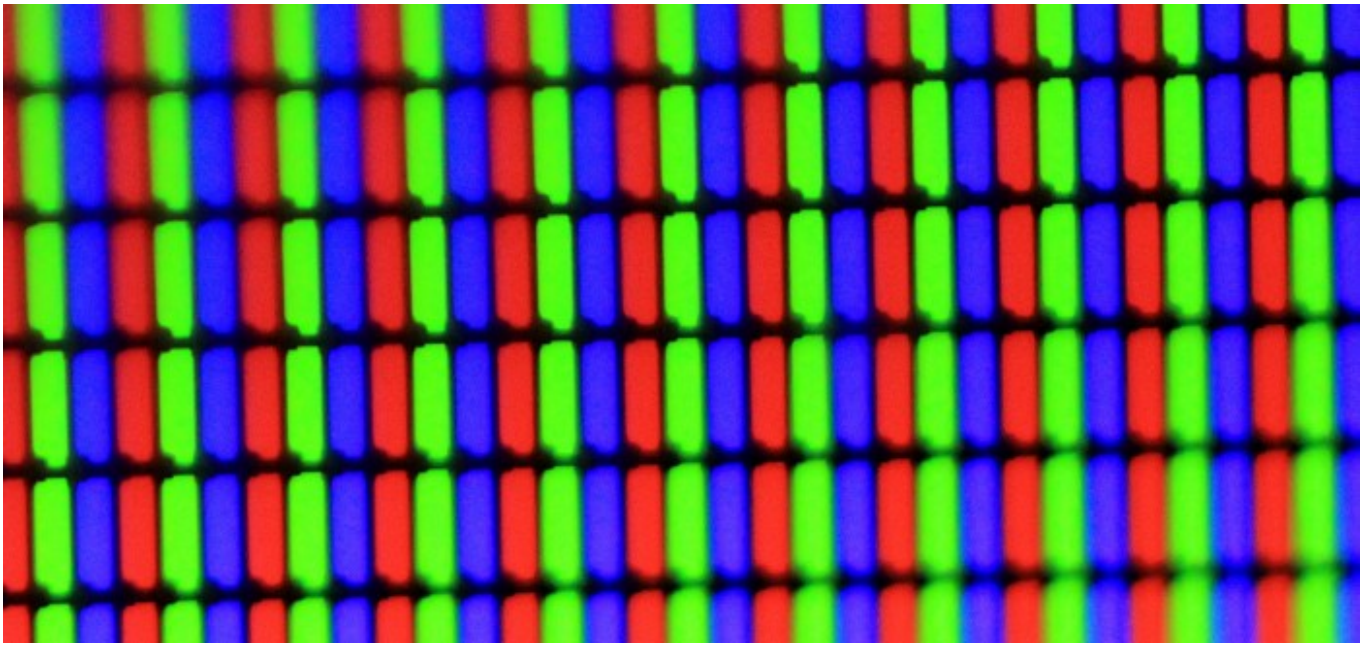
На самом деле модели хранения этих байтов в памяти Python и файле с картинкой бывают очень сложными — например со сжатием. Однако мы будем работать с исходными, «чистыми» данными.

Итак, каждый цвет — это совокупность трёх целых чисел (в Python её можно представить кортежем или списком). Кстати, сумма этих трех чисел говорит о яркости пикселя: чем сумма больше, тем пиксель кажется ярче. На самом деле и тут все сложнее, чем кажется: яркость каждого компонента для глаза не одинакова, однако примем это упрощение.

Например, (0, 0, 0) — это черный цвет. Его яркость минимальна, оттенков нет.

- (255, 255, 255) — белый, максимальная яркость.
- (255, 0, 255) — очень насыщенный пурпурный (красный + синий).
- (255, 255, 0) — ярко-желтый (красный + зеленый).
- (100, 100, 100) — серый.

Красный, зеленый и синий выбраны в качестве основных цветов из-за особенностей цветовой чувствительности рецепторов нашего глаза. Кстати, если мы сильно увеличим матрицу смартфона или монитора, который светит чистым белым светом, то увидим что-то вроде этого:

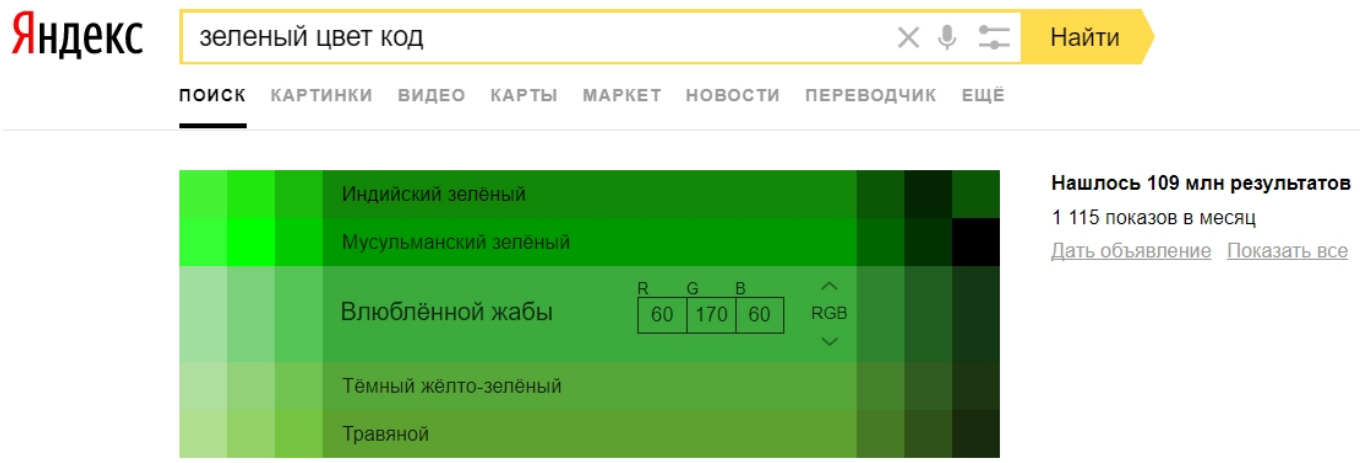


Да-да, это (255, 255, 255).

Итак, для нас изображение — это список списков, элементами которого будут кортежи цвета.

Кстати, легко заметить, что в нашей модели всего 256 x 256 x 256 = 16777216 разных цветов. Этого вполне достаточно, чтобы человеческий глаз не замечал дискретности (конечного числа оттенков) цветовой модели.

У Яндекса есть специальный барабан, который позволяет знакомиться и подбирать цвета и их коды:



PIL. Установка библиотек

Для работы с растровыми изображениями мы будем использовать библиотеку PIL (Python image library), а точнее ее модификацию под названием `Pillow`.

Для установки пакетов в Python служит специальная утилита командной строки **`pip`**, которая является еще и модулем.

Чтобы установить пакет, нужно выполнить команду `pip install`:

```
c:\Python34\Scripts>pip install pillow
Collecting pillow
  Downloading Pillow-4.0.0-cp34-cp34m-win32.whl (1.2MB)
    100% |################################| 1.2MB 485kB/s
Collecting olefile (from pillow)
  Downloading olefile-0.44.zip (74kB)
    100% |################################| 81kB 1.7MB/s
Installing collected packages: olefile, pillow
  Running setup.py install for olefile ... done
Successfully installed olefile-0.44 pillow-4.0.0

c:\Python34\Scripts>
```

Пакет будет скачан с PyPI и установлен.

Кроме опции `install` в `pip` доступны команды:

```
Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible dependencies.
  search                      Search PyPI for packages.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completion.
  help                        Show help for commands.
```

`Pillow` не чисто питоновская библиотека, она написана частично на языке C. Поэтому для некоторых версий Python может потребоваться компиляция кода доступным в системе C-компилятором, потому что `pip` сможет скачать только исходные коды библиотеки. Если такого компилятора нет (такое обычно бывает в windows-системах), стоит поискать скомпилированные версии в интернете (готовые к установке файлы имеют расширение `.whl`). Например, множество популярных библиотек можно найти на странице (http://www.lfd.uci.edu/~gohlke/pythonlibs/) сайта лаборатории флуоресцентной динамики Калифорнийского университета.

Также чтобы не задумываться о сложностях при установке библиотек, можно установить дистрибутив Anaconda (https://www.continuum.io/downloads). В нём есть все необходимые библиотеки Python и не только они.

Модельный пример

Рассмотрим пример работы с изображением, в котором мы:

1. Пройдем по каждому пикселю в изображении.
2. Получим для него значение цвета в RGB-нотации.
3. Присвоим этому пикселю новое значение цвета (поменяем составляющие).
4. В конце сохраним получившееся изображение с новым именем.

Начальное изображение в этом примере никак не меняется, но от него можно отталкиваться в дальнейшей работе.

Итак, приступим.

In [1]:

```
from PIL import Image

im = Image.open("images/Риана.jpg")
pixels = im.load() # список с пикселями
x, y = im.size # ширина (x) и высота (y) изображения

for i in range(x):
    for j in range(y):
        r, g, b = pixels[i, j]
        pixels[i, j] = g, b, r

im.save("images/Риана2.jpg")
```

Для работы с изображением нам нужен объект **Image**, который находится в библиотеке PIL (пишется большими буквами).

Мы открываем изображение с диска функцией **open**, а потом получаем список пикселей этого изображения, используя функцию **load**.

Обратите внимание: `pixels` устроен так, что индексация в нем идет кортежами, поэтому есть запись `pixels[i, j]`, а не `pixels[i][j]`, что возможно, было бы удобней и привычней. Это особенность библиотеки: создателям показалось, что так будет архитектурно уместнее.

Мы используем множественное присваивание, поэтому пишем:

```
r, g, b = pixels[i, j]
```

вместо

```
pixel = pixels[i, j]
r = pixel[0]
g = pixel[1]
b = pixel[2]
```

Множественное присваивание позволяет писать более простой и лаконичный код. Именно так мы поступили и в случае с вычислением x и y.

Фильтры

Когда-то Instagram превратился из заурядной социальной сети в очень популярный феномен именно из-за удачной реализации встроенных фильтров. Фильтры можно было накладывать на фото — фотографии после этого обычно становились красивыми и похожими на профессиональные.

Фильтр можно воспринимать как любое преобразование заданного изображения. Чтобы добиться наилучшего эффекта, их можно накладывать последовательно.

Фильтры очень широко применяются в киноиндустрии. Сравните цветовую гамму молодежных комедий или современных блокбастеров с, например, классическим «Шерлоком Холмсом».

Иначе говоря, фильтры очень востребованы — начиная от самых простых и заканчивая работами с привлечением искусственного интеллекта, например, в проекте Prisma (http://prisma-ai.com/).

В библиотеке PIL реализовано много встроенных фильтров и инструментов (вырезание, изменение размеров, и т. д.). Фактически это такой программируемый мини-Photoshop, но мы попытаемся поработать с фильтрами самостоятельно, чтобы поучиться восприятию цветовой палитры и алгоритмизации.

Для начала попробуем превратить изображение в черно-белое. Черно-белое изображение содержит только информацию о яркости, но не о цветах. У таких изображений все три компоненты имеют одинаковое значение, поэтому мы можем просто «размазать» суммарную яркость пикселя поровну по трём компонентам:

```
for i in range(x):
    for j in range(y):
        r, g, b = pixels[i, j]
        bw = (r + g + b) // 3
        pixels[i, j] = bw, bw, bw
```

Можно сказать, что мы «слили» содержимое контейнеров R, G, B в одну ёмкость, а затем разлили обратно — но уже поровну в каждый контейнер. Суммарная яркость пикселя осталась прежней, но информация о цвете не сохранилась. Фотография же стала более «задумчивой».

Попробуем поменять местами зелёный и синий каналы:

```
for i in range(x):
    for j in range(y):
        r, g, b = pixels[i, j]
        pixels[i, j] = r, b, g
```

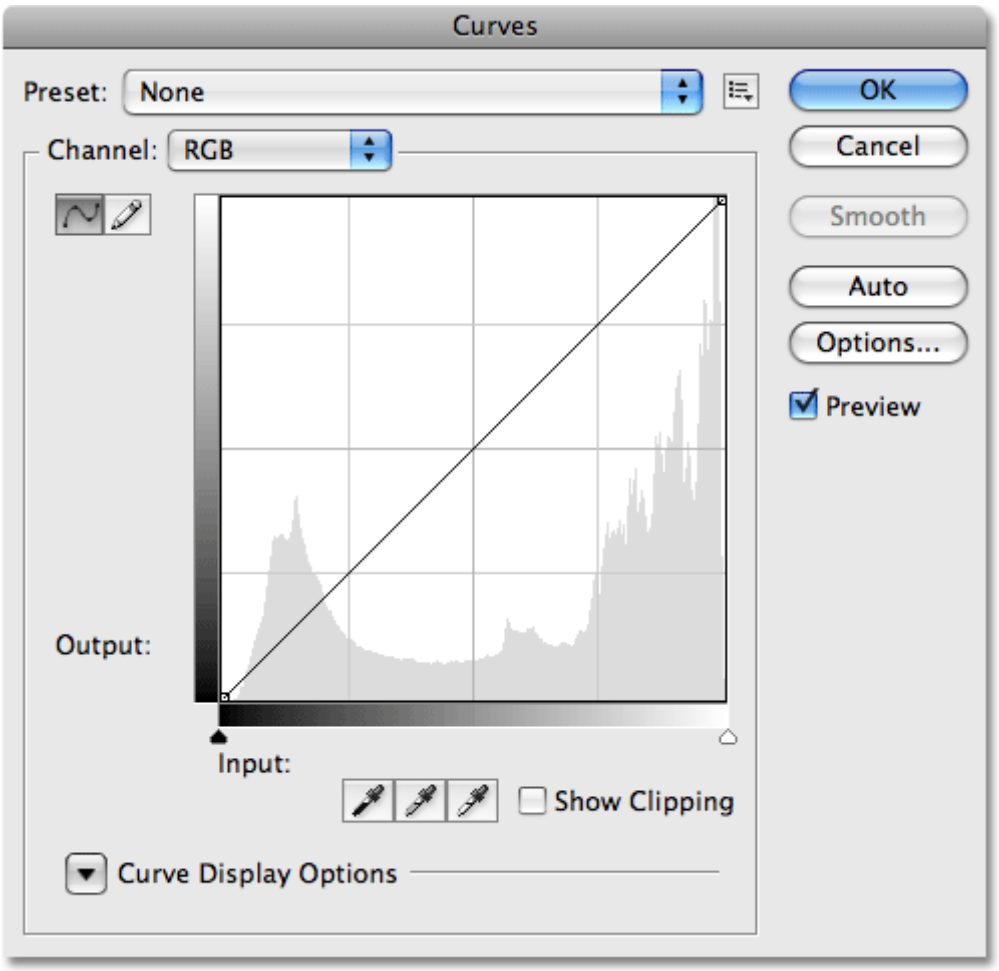
Давайте подумаем над тем, как получить негатив. Если на позитиве белое изображение (255), то на негативе должно быть черное (0) и наоборот. То есть для значения x негативом будет 255 - x.

```
for i in range(x):
    for j in range(y):
        r, g, b = pixels[i, j]
        pixels[i, j] = 255-r, 255-g, 255-b
```

Как видим, на негативе можно рассмотреть некоторые детали, которые не видны на позитиве.

Во многих редакторах (включая Photoshop) есть инструмент Кривые (Curves).

Он позволяет задать функцию, меняющую яркость всего пикселя или отдельной компоненты в зависимости от исходной яркости. Изначально эта функция представляет собой прямую y=x.



В Python можно написать функцию, которая работает как инструмент Curves. Например, мы можем высветлить темные участки в изображении, не трогая светлые. Это очень частая операция: например, когда на снимке светлое небо и очень темное здание, потому что фотоаппарат подстроился под яркость неба.

«Высветлить» означает увеличить значения всех цветовых компонентов на какой-то коэффициент. Важно помнить, что эти значения не могут быть больше 255.

```
def curve(pixel):
    r, g, b = pixel
    brightness = r + g + b
    if brightness < 60:
        k = 60 / brightness
        return min(255, int(r * k ** 2)), min(255, int(g * k ** 2)), min(255, int(b* k
** 2))
    else:
        return r, g, b

for i in range(x):
    for j in range(y):
        pixels[i, j] = curve(pixels[i, j])
```

Результат:

А теперь давайте немного поработаем с изображениями.