* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/123/groups/568)

[Урок Библиотеки №2](https://lyceum.yandex.ru/courses/123/groups/568/lessons/645)

**Библиотеки Python. Часть № 2 (графика)**

**План урока**

1

[Обработка изображений](https://lyceum.yandex.ru/courses/123/groups/568/lessons/645/materials/1452#1)

2

[Растровые изображения](https://lyceum.yandex.ru/courses/123/groups/568/lessons/645/materials/1452#2)

3

[PIL. Установка библиотек](https://lyceum.yandex.ru/courses/123/groups/568/lessons/645/materials/1452#3)

4

[Модельный пример](https://lyceum.yandex.ru/courses/123/groups/568/lessons/645/materials/1452#4)

**Аннотация**

*В первом уроке, посвящённом 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.*

**1. Обработка изображений**

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 и Pinterest.

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. 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.

**2. 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 Риана).



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).

**RGB**

В модели RGB каждый из цветов представляется совокупность трёх компонентов: красного, синего и зелёного. Значение каждого компонента лежит в диапазоне от 0 (минимум) до 255 (максимум), занимая таким образом 1 байт в памяти.

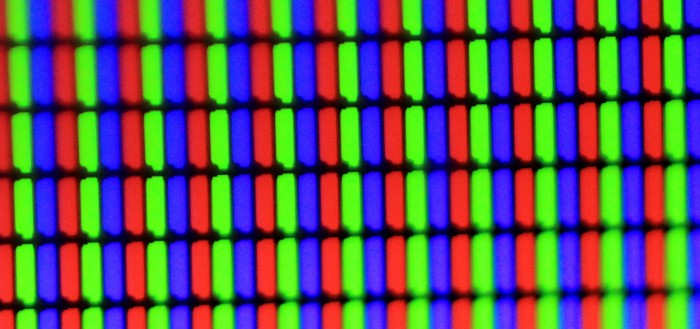
На самом деле модели хранения этих байтов в памяти Python и файле с картинкой бывают очень сложными — например, со сжатием. Однако мы будем работать с исходными, «чистыми» данными.

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

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

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

Красный, зелёный 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:

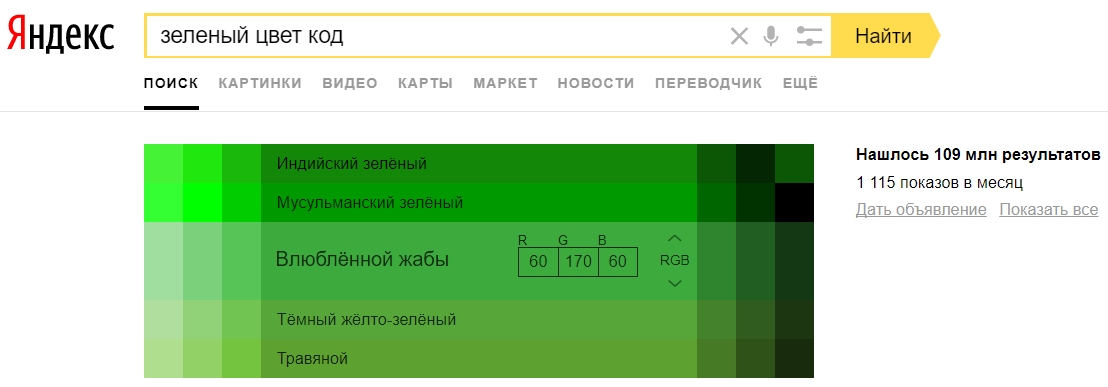


Да-да, это (255, 255, 255).

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

Кстати, легко заметить, что в нашей модели всего 256×256×256 = **16 777 216** 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:



**3. PIL. Установка библиотек**

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

**PIP**

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

Чтобы установить пакет, нужно выполнить команду pip install <Имя модуля>. Пакет будет скачан с PyPI и установлен, вы увидите примерно следующее:

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>

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 С. Поэтому для некоторых версий Python может потребоваться компиляция кода доступным в системе С-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.

**4. Модельный пример**

Рассмотрим пример работы с изображением, в котором мы:

1. Пройдём 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.

Для работы нам потребуется файл с изображением — Риана.jpg, который нужно сохранить в тот же каталог, где будут лежать программы по его обработке.

**from** PIL **import** Image

im = Image.open("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("Риана2.jpg")

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

**Функция open**

Мы открываем изображение с диска функцией open. В функции open в скобках указывается или абсолютный путь к файлу или просто имя файла, если файл размещён в том же каталоге, где и сама программа.

Потом получаем список пикселей этого изображения, используя функцию **load**. Функцию применяем к объекту, загруженному в переменную **im**. После применения функции — получаем двумерный список, где для каждого пикселя хранится кортеж — цвет пикселя в палитре RGB.

**pixels**

Обратите внимание: pixels устроен так, что индексация в нём идёт 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.

Функцией **size**, применённой к объекту im, мы получаем размер изображения: сначала ширину, потом высоту изображения в пикселях, что соответствует размерности pixels.

Далее переберем все элементы pixels (двумя циклами for) и для каждого элемента получим значение трёх компонент цвета и запишем в массив pixels эти значения, но изменив порядок значений.

Для получения трёх компонентов цвета каждого пикселя 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 у.

Затем при помощи функции **save** сохраняем изменённый список пикселей в изображение с именем Риана2.jpg.

**Важно**

Обратите внимание, что в данном случае появляется новая картинка, в том же месте, где находилась начальная. Начальное изображение осталось без изменений, а новое изображение получено из начального путём изменения значений цветовых компонент для каждого пикселя.

[Справка](https://yandex.ru/support/lyceum-students)

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Яндекс.Лицей», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты