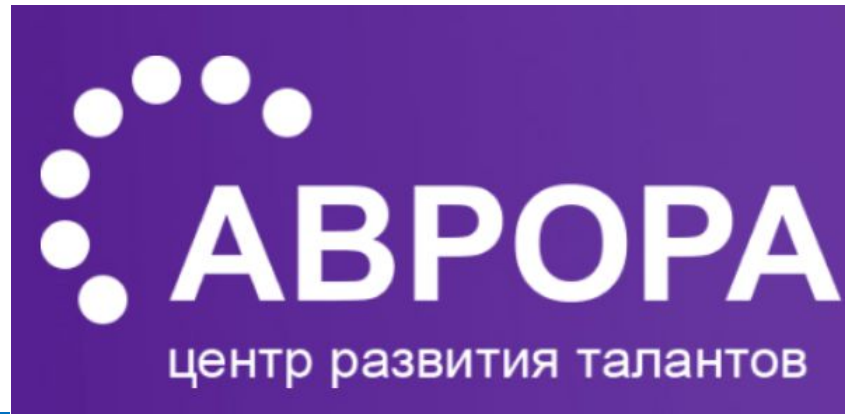
Лицей Академии Яндекса



Основы программирования на языке Python

ПОЛУПАНОВ Дмитрий Васильевич

к.т.н., доцент

А на прошлом уроке было…

Разбор ошибок и задания на закрепление прошедшего материала

Python Imaging Library (PIL)

В настоящее время актуальный модуль – pillow, модификация библиотеки PIL

Официальная страница поддержки

http://www.pythonware.com/products/pil/

Возможности библиотеки:

- поддержка бинарных, полутоновых, индексированных, полноцветных и CMYK изображений;
- поддержка форматов BMP, EPS, GIF, JPEG, PDF, PNG, PNM, TIFF и некоторых других на чтение и запись;
- поддержка множества форматов (ICO, MPEG, PCX, PSD, WMF и др.) только для чтения;
- преобразование изображений из одного формата в другой;
- правка изображений (использование различных фильтров, масштабирование, рисование, матричные операции и т. д.);
- использование библиотеки из Tkinter и PyQt.

Делаем отражения изображения

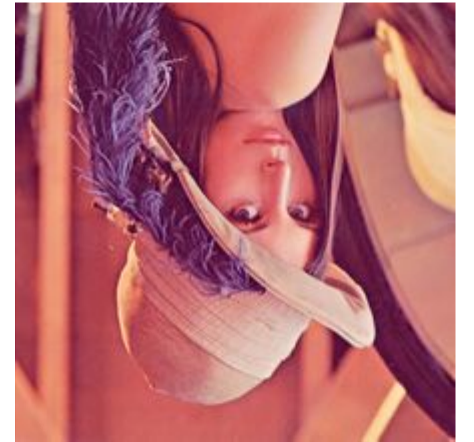
Возьмем в качестве примера стандартное тестовое изображение «Лена»

Вертикальное

```
for i in range(x // 2):
    for j in range(y):
        pixels[i, j], pixels[x - i - 1, j] \
            = pixels[x - i - 1, j], pixels[i, j]
```

Горизонтальное отражение

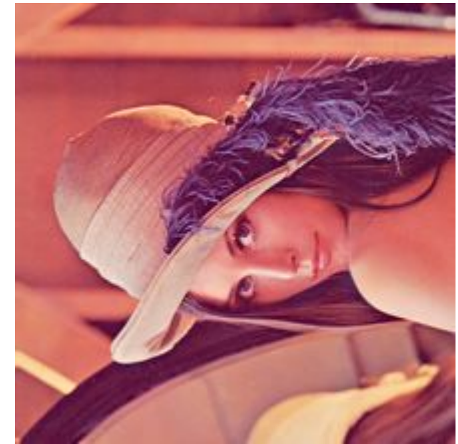
```
for i in range(x):
    for j in range(y // 2):
        pixels[i, j], pixels[i, y - j - 1] \
            = pixels[i, y - j - 1], pixels[i, j]
```



Отражение относительно главной диагонали

Искомое отображение должно быть квадратным, с одинаковой шириной и высотой

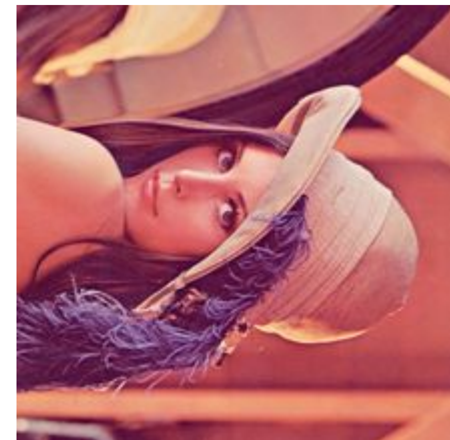
```
for i in range(x):
    for j in range(i):
        pixels[i, j], pixels[j, i] \
            = pixels[j, i], pixels[i, j]
```



Отражение относительно побочной диагонали

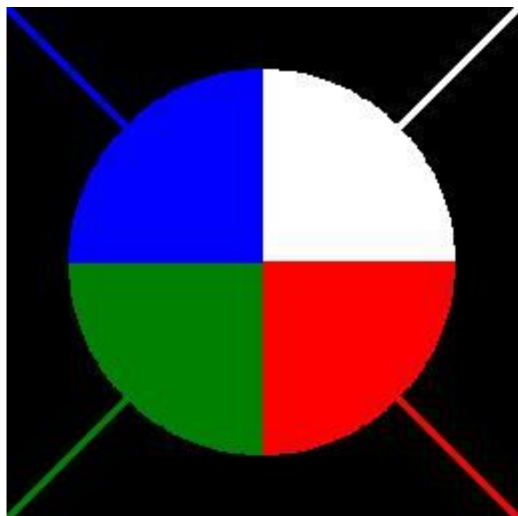
Искомое отображение должно быть квадратным, с одинаковой шириной и высотой

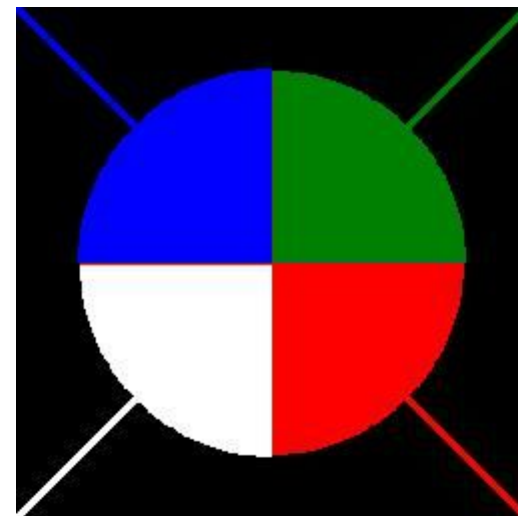
```
for i in range(x):
    for j in range(x - i):
        pixels[j, i], pixels[x - i - 1, x - j - 1] \
            = pixels[x - i - 1, x - j - 1], pixels[j, i]
```



Некоторые примеры

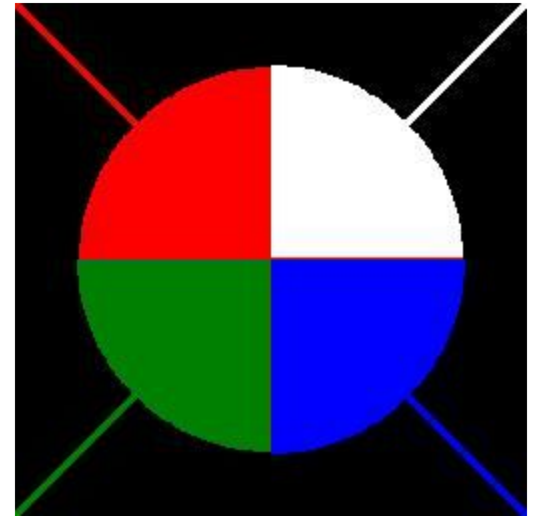
Напишите функцию twist_image(input_file_name, output_file_name), которая будет менять местами правую верхнюю и левую нижнюю четверти изображения.
Параметр input_file_name задаёт имя исходного файла, а output_file_name — имя файла, куда следует сохранить результат.





```
from PIL import Image


def twist_image(input_file_name, output_file_name):
    im = Image.open(input_file_name)
    pixels = im.load()
    x, y = im.size
    for i in range(x // 2, x):
        for j in range(y // 2):
            pixels[i, j], pixels[x - i - 1, y - j - 1]  \
                = pixels[x - i - 1, y - j - 1], pixels[i, j]
    im.save(output_file_name)
```

А теперь будем менять местами правую нижнюю и левую верхнюю четверти изображения



```python
from PIL import Image


def twist_image(input_file_name, output_file_name):
    im = Image.open(input_file_name)
    pixels = im.load()
    x, y = im.size
    for i in range(x // 2, x):
        for j in range(y // 2, y):
            pixels[i, j], pixels[x - i - 1, y - j - 1]  \
                = pixels[x - i - 1, y - j - 1], pixels[i, j]
    im.save(output_file_name)
```

А если хочется что-нибудь самому нарисовать?

Для рисования на изображении используется объект `Draw` из библиотеки `ImageDraw`. У этого объекта есть много инструментов для создания графических примитивов: прямых, кривых, точек, прямоугольников, дуг и т.д.

```
new_image = Image.new('RGB', (x_size, y_size), (r, g, b))
```

- `Image.new` — Функция создания «холста»
- `'RGB'` — Палитра
- `(x_size, y_size)` — Размеры по осям x и y
- `(r, g, b)` — Код цвета в палитре

Подробно про библиотеку ImageDraw:
https://pillow.readthedocs.io/en/stable/reference/ImageDraw.html#functions

```
draw = ImageDraw.Draw(new_image)
```

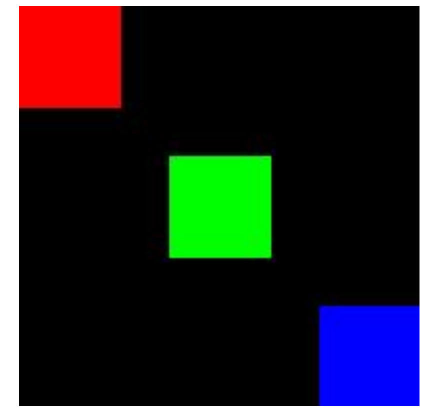
На холсте создаем место, где будет создан рисунок

На черном прямоугольнике размера 100*200 создать красную линию, проходящую из левого верхнего в правый нижний угол, толщиной 1

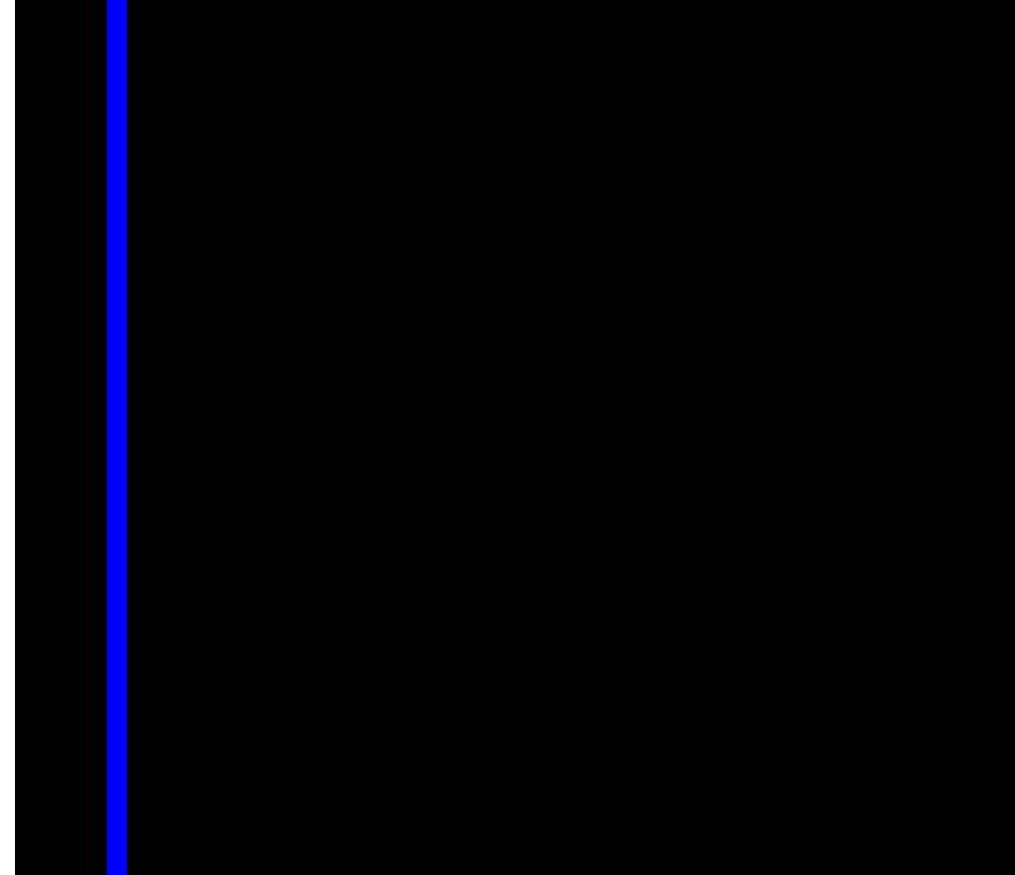
```
from PIL import Image, ImageDraw


new_image = Image.new("RGB", (100, 200), (0, 0, 0))
draw = ImageDraw.Draw(new_image)
draw.line((0, 0, 100, 200), fill=(255, 0, 0), width=1)
# сохраним изображением в файл формата PNG
new_image.save('line.png', 'PNG')
```

На черном квадрате размерность 200* 200 разместить три квадрата размера 50*50 – красный, начинающийся в левом верхнем углу, зеленый – посередине и синий, заканчивающийся в правом нижнем углу

```
from PIL import Image, ImageDraw


new_image = Image.new("RGB", (200, 200), (0, 0, 0))
draw = ImageDraw.Draw(new_image)
draw.rectangle((0, 0, 50, 50), fill=(255, 0, 0))
draw.rectangle((75, 75, 125, 125), fill=(0, 255, 0))
draw.rectangle((150, 150, 200, 200), fill=(0, 0, 255))
new_image.save('rectangle.png', 'PNG')
```



На черном фоне нарисуем синюю вертикальную линию со сдвигом 50 от левого края. Холст – квадрат 500*500, линия толщиной 10



```
from PIL import Image, ImageDraw

new_image = Image.new("RGB", (500, 500), (0, 0, 0))
draw = ImageDraw.Draw(new_image)
color = 0, 0, 255
draw.line((50, 0, 50, 500), fill=color, width=10)
new_image.save('line.png', 'PNG')
```

Нарисуем дугу между начальными и конечными углами, внутри данной ограничительной области

```
from PIL import Image, ImageDraw

new_image = Image.new("RGB", (100, 100), (0, 0, 0))
draw = ImageDraw.Draw(new_image)
color = 0, 0, 255
draw.arc((10, 10, 90, 90), -45, 90, fill=color, width=10)
new_image.save('arc.png', 'PNG')
```

? Какого цвета дуга?

Рисуем многоугольник

```
from PIL import Image, ImageDraw

new_image = Image.new("RGB", (600, 200), (0, 0, 0))
draw = ImageDraw.Draw(new_image)
color = 255, 0, 255
draw.polygon((10, 10, 500, 150, 20, 200),
             fill='white', outline=color)
new_image.save('poligon.png', 'PNG')
```

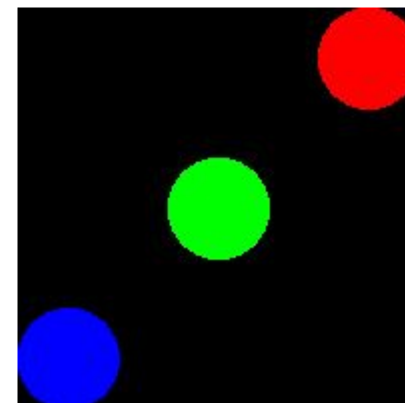
Координаты вершин

Цвет фона

Цвет границы

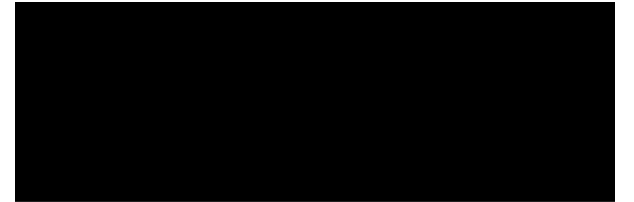
На черном квадрате размерность 200* 200 разместить три круга размера 50*50 – красный, начинающийся в правом верхнем углу, зеленый – посередине и синий, заканчивающийся в левом нижнем углу

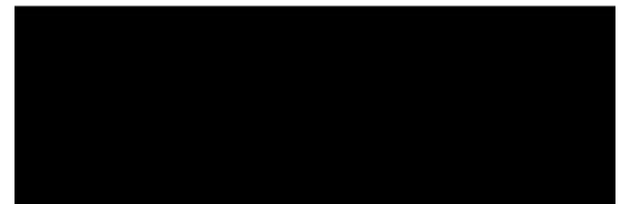
```
from PIL import Image, ImageDraw

new_image = Image.new("RGB", (200, 200), (0, 0, 0))
draw = ImageDraw.Draw(new_image)
draw.ellipse((150, 0, 200, 50), fill=(255, 0, 0))
draw.ellipse((75, 75, 125, 125), fill=(0, 255, 0))
draw.ellipse((0, 150, 50, 200), fill=(0, 0, 255))
new_image.save('circule.png', 'PNG')
```



По середине черного квадрата размера 300*300 сделать горизонтальную полосу толщиной 100

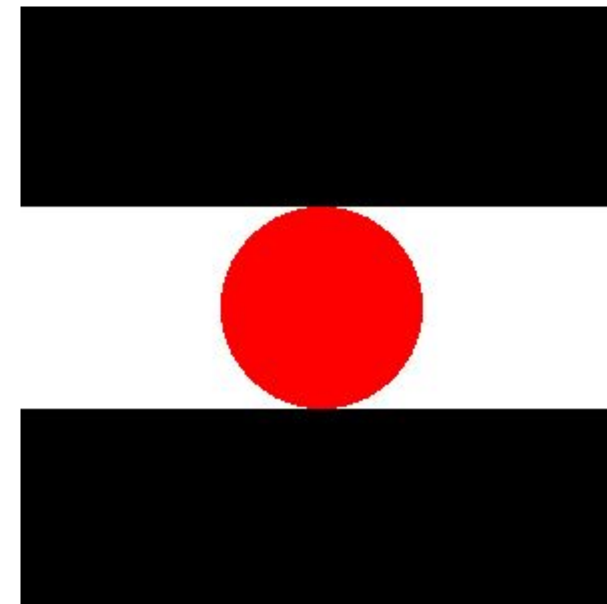
```
from PIL import Image, ImageDraw

new_image = Image.new("RGB", (300, 300), (0, 0, 0))
draw = ImageDraw.Draw(new_image)
draw.rectangle((0, 100, 300, 200), fill=(255, 255, 255))
new_image.save('rectangle.png', 'PNG')
```





В середине предыдущего рисунка разместить красный круг

```
from PIL import Image, ImageDraw

new_image = Image.new("RGB", (300, 300), (0, 0, 0))
draw = ImageDraw.Draw(new_image)
draw.rectangle((0, 100, 300, 200), fill=(255, 255, 255))
draw.ellipse((100, 100, 200, 200), fill=(255, 0, 0))
new_image.save('rectangle1.png', 'PNG')
```



Шахматная доска

Не нужно создавать черные квадраты, поскольку исходное поле у нас – черное. Нужно сделать только белые квадраты

Импортируем из библиотеки необходимые объекты

```
from PIL import Image, ImageDraw
```

Объявим функцию, аргументы которой - количество клеток по горизонтали (и по вертикали) и размер клетки.
Внутри функции создадим новое изображение соответствующего размера, на черном фоне

```
def board(num, size):
    new_image = Image.new("RGB", (num * size, num * size), (0, 0, 0))
    draw = ImageDraw.Draw(new_image)
```

```
for x in range(num):
    for y in range((x + 1) % 2, num, 2):
        draw.rectangle((x * size, y * size,
                        (x + 1) * size - 1,
                        (y + 1) * size - 1),
                       fill=(255, 255, 255))
```

Сохраняем получившееся изображение в файле

```
new_image.save('res.png', "PNG")
```

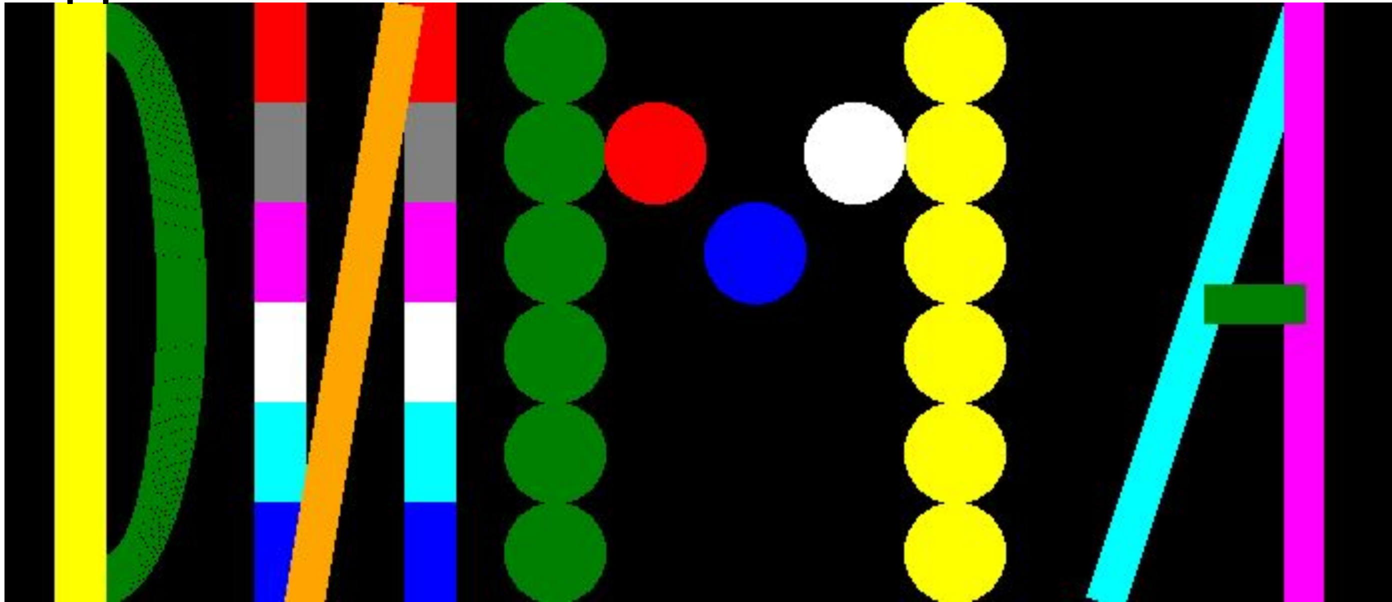
Красивое имя

Напишите программу, которая красиво рисует имя своего автора. Каждую букву надо нарисовать с помощью графических примитивов (линий, дуг и прочее), для заливки так же необходимо использовать функции библиотеки PIL.

Главное правило – не использовать встроенные шрифты.

Сохраните полученный рисунок в файле name.png.

В качестве примера – имя «Дима»



Импортируем из библиотеки необходимые элементы.
Создаем основу – черное поле размера 700 на 300

```
from PIL import Image, ImageDraw

new_image = Image.new('RGB', (700, 300), (0, 0, 0))
```

Желтый прямоугольник

```
draw.rectangle((25, 0, 50, 300), 'yellow')
```

Зеленая дуга

```
draw.arc(((0, 0), (100, 300)), -90, 90, 'green', 25)
```

Две вертикальных палочки в букве «И». Сделаем их разного цвета

```
color = ['red', 'gray', 'magenta', 'white', 'cyan', 'blue']
for i in range(6):
    draw.rectangle((125, i * 50, 150, i * 50 + 50), color[i])
    draw.rectangle((200, i * 50, 225, i * 50 + 50), color[i])
```

Перекладина буквы «И». Оранжевая

```
draw.line(((150, 300), (200, 0)), 'orange', 20)
```

Букву «М» будем делать с помощью кружков разного цвета

Для этого используем объект эллипс `draw.ellipse`

Две вертикальных палочки в букве «М».

```python
for i in range(0, 300, 50):
    draw.ellipse(((250, i), (300, i + 50)), 'green')
    draw.ellipse(((450, i), (500, i + 50)), 'yellow')
```

Середина буквы «М»

```python
draw.ellipse(((300, 50), (350, 100)), 'red')
draw.ellipse(((400, 50), (450, 100)), 'white')
draw.ellipse(((350, 100), (400, 150)), 'blue')
```

Букву «А» нарисуем с помощью трех линий

```
draw.line(((550, 300), (650, 0)), 'cyan', 20)
draw.line(((650, 300), (650, 0)), 'magenta', 20)
draw.line(((600, 150), (650, 150)), 'green', 20)
```

И сохраним получившийся результат

```
new_image.save('name.png')
```

Библиотеки. Часть № 3 (графика + звук)

Применение фильтров к изображению.

Модуль wave для работы со звуком

Фильтры

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

Давайте применим фильтры к нашей старой знакомой Риане. Искомое изображение находится в папке с программами и называется image.jpg

Сделаем изображение черно-белым

```
from PIL import Image

im = Image.open('image.jpg')
pixels = im.load()
x, y = im.size

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

im.save('res.jpg')
```

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.

Поменяем 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
```

А как получить негатив?

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
```

Высветлить компонент

Во многих графических 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.

Давайте высветлим темные участки в изображении, не трогая светлые. Для этого напишем функцию, которая работает как инструмент Curves

«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
```

Изменения в теле программы

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

Работа со звуком. Модуль wave

Фильтры можно накладывать не только на изображение, но и на звуковые файлы. Для манипуляции с «сырыми», необработанными аудиоданными предназначен модуль `wave`.

Звуковые данные хранятся в файлах с расширением `.wav`.

Сырые аудиоданные представляет собой зависимость амплитуды звукового сигнала от времени. Вдоль оси абсцисс откладывается время, вдоль оси ординат — амплитуда (интенсивность, громкость) звукового сигнала.

График строится по точкам, причём вместо пар (t, y) хранятся только значения y, а ось абсцисс задана частотой дискретизации (количеством отсчетов в секунду) — как правило, она составляет 44 100 Гц.

Представление звукового файла для программиста - это список целых чисел (положительных и отрицательных) — значения амплитуды звукового сигнала.

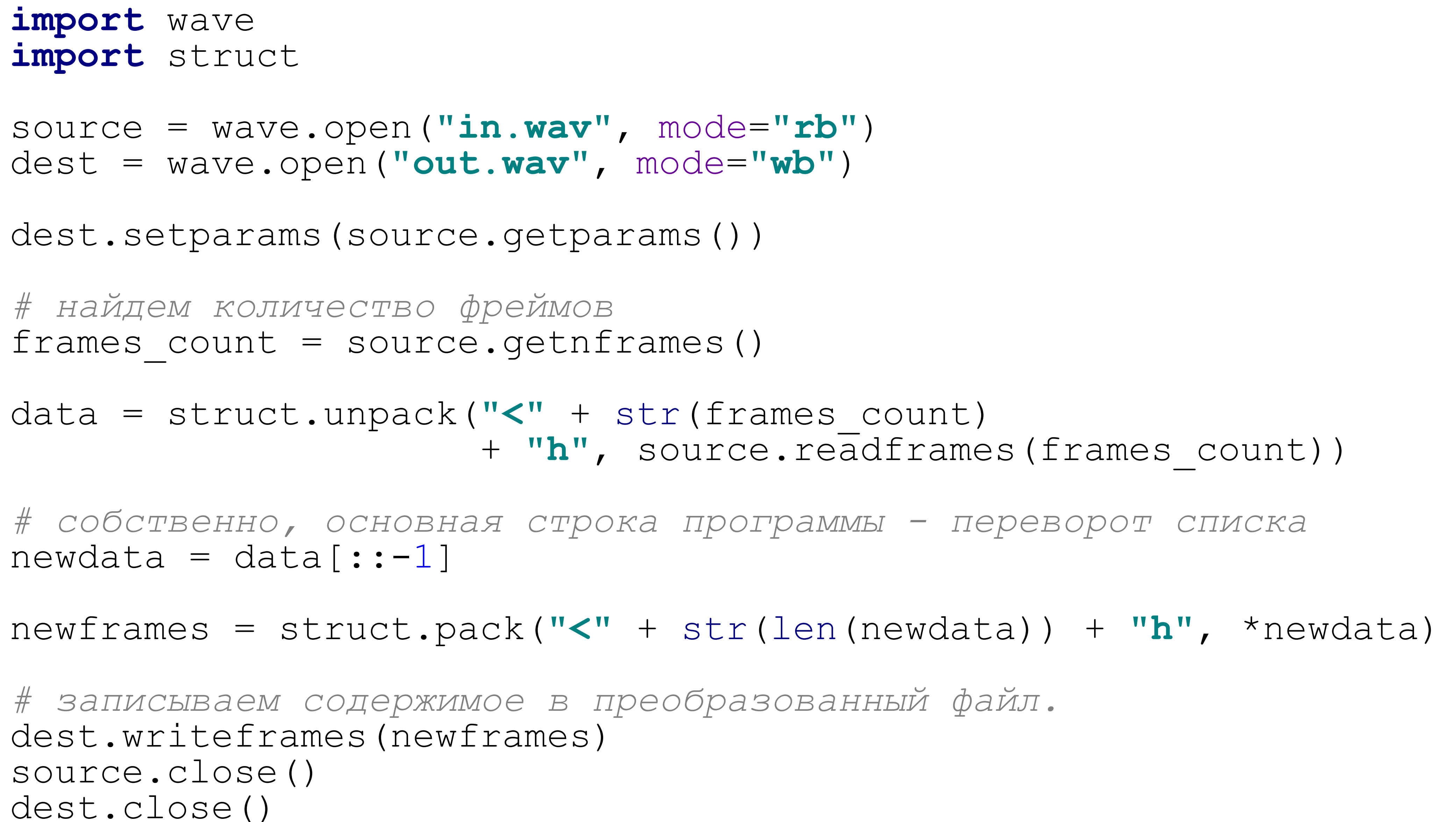
Создадим программу, которая будет разворачивать звуковой файл в обратную сторону, то есть проигрывать задом наперед

Будем открывать два файла: исходный — source и формируемый — dest.

Одно значение амплитуды в терминах библиотеки wave называется фреймом

Используем возможности встроенного модуля struct. Пока договоримся, что функции этого модуля могут на лету распаковывать и запаковывать данные разной природы.

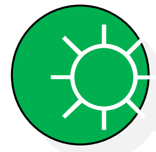
Модуль struct позволяет паковать и извлекать несколько значений в бинарные структуры. Это чем-то похоже на структуры в С. В примере со звуковыми волнами значение y хранится как знаковое двухбайтное число.



```python
import wave
import struct

source = wave.open("in.wav", mode="rb")
dest = wave.open("out.wav", mode="wb")

dest.setparams(source.getparams())

# найдем количество фреймов
frames_count = source.getnframes()

data = struct.unpack("<" + str(frames_count)
                     + "h", source.readframes(frames_count))

# собственно, основная строка программы - переворот списка
newdata = data[::-1]

newframes = struct.pack("<" + str(len(newdata)) + "h", *newdata)

# записываем содержимое в преобразованный файл.
dest.writeframes(newframes)
source.close()
dest.close()
```

Некоторые наблюдения

- Если убрать , например, каждый второй фрейм, то ускорим произведение вдвое. При этом частота тоже вырастет в два раза.
- Если увеличить все фреймы в какое-то количество раз, то произведение станет громче, а если уменьшим — тише.
- Копируя каждый фрейм 2 раза, замедлим воспроизведение и понизим частоту.



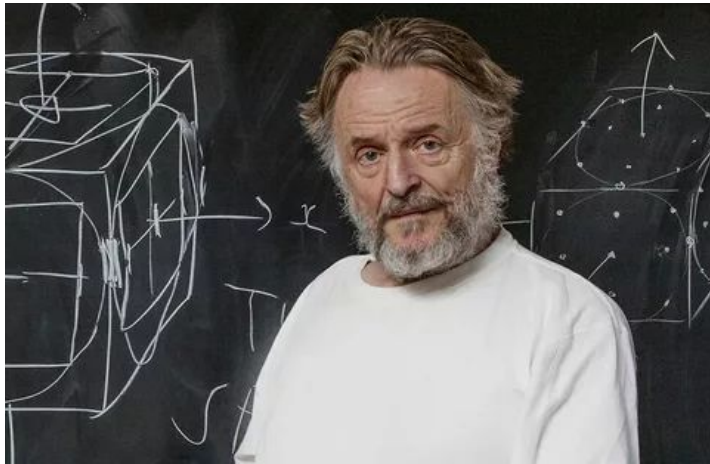
На телевидении и радио часто ускоряют видео и аудио на 5-10%: это незаметно для уха, но позволяет разместить больше рекламы в эфире.

Библиотеки Python. Numpy. Часть 3

Клеточные автоматы

Клеточные автоматы. Игра «Жизнь»

Клеточный автомат — это модель однородного пространства с некоторыми клетками. Каждая клетка может находиться в одном из нескольких состояний и иметь некоторое количество соседей. Задаются правила перехода из одного состояния в другое в зависимости от текущего состояния клетки и её соседей



Джон Хортон Конвей (род. 26 декабря 1937)

В 1970 году **Джон Конвей** придумал игру «Жизнь» («Genesis»), основанную на клеточном автомате, ставшую довольно популярной и повлиявшую на назвитие многих точных наук.

Пространство «Жизни» — бесконечное поле клеток.

Каждая клетка имеет 8 соседей (сверху, снизу, справа, слева и по диагонали). Клетка может иметь два состояния: **живая** (на клетке стоит фишка) и **мёртвая** (фишки нет).

Правила

- Если клетка была живой, то она выживет, если у неё 2 или 3 соседа. Если соседей 4, 5, 6, 7 или 8, то она умирает от перенаселённости, а если 0 или 1 — то от одиночества.
- Новая клетка рождается в поле, у которого есть ровно 3 соседа.

Время дискретно и считается поколениями.

Всё начинается с начальной расстановки фишек (0 поколение), в дальнейшем рассматривается эволюция клеточного пространства в 1, 2, 3 и т. д. поколении.

Процессы смерти и рождения происходят одновременно, после чего строится следующее поколение.

Игра прекращается, если:

- на поле не останется ни одной «живой» клетки;
- конфигурация на очередном шаге в точности (без сдвигов и поворотов) повторит себя же на одном из более ранних шагов (складывается периодическая конфигурация);
- при очередном шаге ни одна из клеток не меняет своего состояния (складывается стабильная конфигурация: предыдущее правило, вырожденное до одного шага назад)

Программируем игру «Жизнь»

Заданим поле 10 x 10, в центр которого поместим конструкцию, известную как «глайдер»

```
import numpy as np

population = np.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, 1, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
     [0, 0, 0, 1, 1, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=np.uint8)
```

Поле имеет тип uint8, чтобы оно занимало меньше памяти. Каждый его элемент занимает ровно 1 байт (8 бит) и является целым беззнаковым (unsigned) числом в диапазоне от 0 до 255.

Живые клетки обозначаются единицей, а мёртвые — нулём.

Что делать на границах поля?

Невозможно обеспечить бесконечность в обоих направлениях, поэтому замкнём поле само на себя. Если выйти за нижнюю границу, мы окажемся наверху, а если за правую — появимся слева, и наоборот. Получается что-то вроде глобуса.

В numpy есть операция `roll` для массивов. Она сдвигает исходный массив вдоль одного из измерений (в данном случае — строки или столбца)

Например, вот так можно сдвинуть исходный массив на две строки вниз

```
new_generation = np.roll(population, 2, 0)
```

Было

```
[[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 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0]
 [0 0 0 1 1 1 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 0 0 0 0 0 0 0 0]]
```

Стало

```
[[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 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0]
 [0 0 0 1 1 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

А куда сдвинемся с помощью такого кода?

```
new_generation = np.roll(population, 2, 1)
```

Правильно, на два столбца вправо

Было

```
[[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 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0]
 [0 0 0 1 1 1 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 0 0 0 0 0 0 0 0]]
```

Стало

```
[[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 1 0 0 0]
 [0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 1 1 1 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 0 0 0 0 0 0]]
```

Можем посчитать количество соседей у каждой клетки, просто сделав 8 копий со сдвигом массива и просуммировав их

```
neighbors = sum([
    np.roll(np.roll(population, -1, 1), 1, 0),
    np.roll(np.roll(population, 1, 1), -1, 0),
    np.roll(np.roll(population, 1, 1), 1, 0),
    np.roll(np.roll(population, -1, 1), -1, 0),
    np.roll(population, 1, 1),
    np.roll(population, -1, 1),
    np.roll(population, 1, 0),
    np.roll(population, -1, 0)
])
```

Матрица соседей

```
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 0 0 0 0]
 [0 0 0 1 1 2 1 0 0 0]
 [0 0 1 3 5 3 2 0 0 0]
 [0 0 1 1 3 2 2 0 0 0]
 [0 0 1 2 3 2 1 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]]
```

Генерируем новую популяцию

Выполним на матрице следующую операцию: «если у клетки 3 соседа, то в следующем поколении на этом месте будет клетка; а если 2 соседа, то клетка будет при условии, что она была "жива" в текущем поколении».

Для этого воспользуемся операторами | (или) и & (и).

Имеется три соседа

```
neighbors == 3
```

```
[[False False False False False False False False False False]
 [False False False False False False False False False False]
 [False False False False False False False False False False]
 [False False False False False False False False False False]
 [False False False  True False  True False False False False]
 [False False False False  True False False False False False]
 [False False False False  True False False False False False]
 [False False False False False False False False False False]
 [False False False False False False False False False False]
 [False False False False False False False False False False]]
```

```
population & (neighbors == 2)
```

```
[[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 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 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 0 0 0 0 0 0 0 0]]
```

Эти примеры - для иллюстрации

Объединим эти два условия

```
population = (neighbors == 3) | (population & (neighbors == 2))
```

```
[[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 0 0 0 0]
 [0 0 0 1 0 1 0 0 0 0]
 [0 0 0 0 1 1 0 0 0 0]
 [0 0 0 0 1 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]]
```

Объединить матрицы с логическими и целочисленными элементами можно, поскольку они в данном случае могут быть сведены друг к другу: 0 — False, 1 — True

Проследим эволюцию глайдера на протяжении 4 поколений. Для этого создадим функцию `next_population()`

```
def next_population(population):
    neighbors = sum([
        np.roll(np.roll(population, -1, 1), 1, 0),
        np.roll(np.roll(population, 1, 1), -1, 0),
        np.roll(np.roll(population, 1, 1), 1, 0),
        np.roll(np.roll(population, -1, 1), -1, 0),
        np.roll(population, 1, 1),
        np.roll(population, -1, 1),
        np.roll(population, 1, 0),
        np.roll(population, -1, 0)
    ])
    return (neighbors == 3) | (population & (neighbors == 2))


for _ in range(4):
    print(population, '\n')
    population = next_population(population)
```

```
[[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 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0]
 [0 0 0 1 1 1 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 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 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 1 0 0 0 0]
 [0 0 0 0 1 1 0 0 0 0]
 [0 0 0 0 1 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]]
```

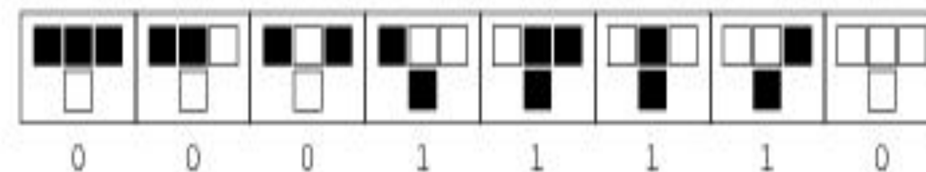
Глайдер «летит»: каждые четыре поколения он сдвигается вниз и вправо. Иными словами, он движется в правый нижний угол

```
[[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 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0]
 [0 0 0 1 0 1 0 0 0 0]
 [0 0 0 0 1 1 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]]
```

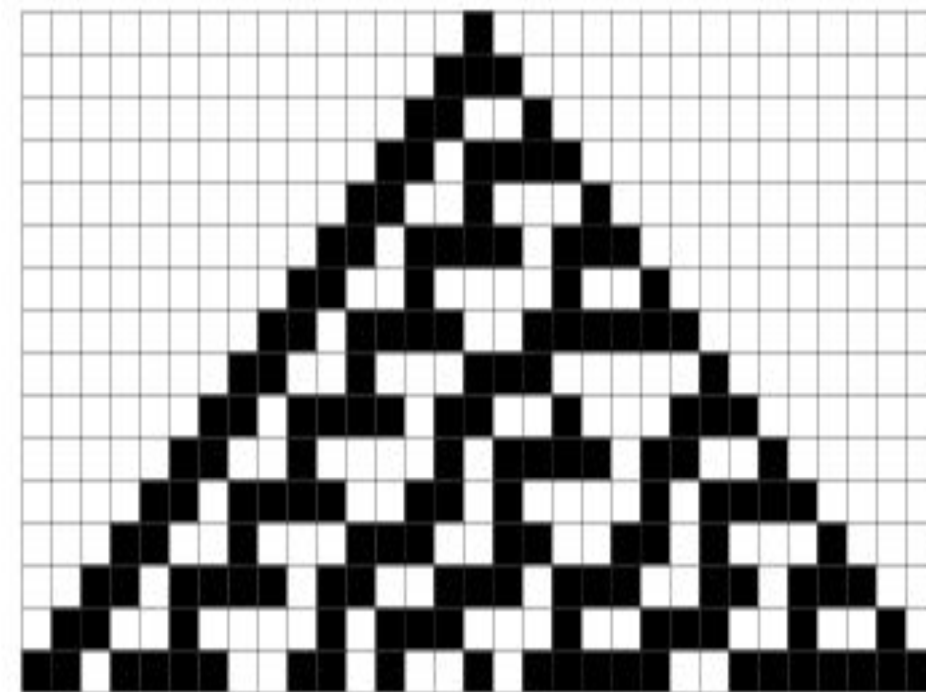
```
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 1 1 0 0 0]
 [0 0 0 0 1 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

Задание numpy. Одномерный клеточный автомат

Ограничение времени 100 секунд
Ограничение памяти 64Mb
Ввод стандартный ввод или input.txt
Вывод стандартный вывод или output.txt



На рисунке показано несколько поколений одномерного клеточного автомата. У каждой клетки только два соседа — слева и справа, а поле представляет собой бесконечную полосу клеток (правый сосед последней клетки – это первая клетка, а левый сосед первой клетки – последняя). Эволюция идёт сверху вниз: первое поколение приведено в верхней строчке, второе — во второй



Правило эволюции дано в верхней части рисунка. Цвет каждой ячейки текущего поколения определяется сочетанием цветов трёх соседних ячеек предыдущего поколения, расположенных над ней: непосредственно сверху и по диагоналям (сверху-слева и сверху-справа).

Всего возможны восемь разных «триплетов», порождающих в следующем поколении либо закрашенную ячейку (1), либо пустую (0). Это дает 256 возможных правил эволюции. В этой задаче используется правило номер 30.

Используя библиотеку numpy, напишите функцию generation(line), которая вычисляет и возвращает десятое поколение клеточного автомата по правилу 30.

Подсказка

Импортируем библиотеку под псевдонимом np

```
import numpy as np
```

Правила клеточного автомата оформим в виде словаря. Ключом будет кортеж, состоящий из клеток - соседей предыдущего поколения, элементом – текущее состояние клетки. 0 – белая клетка, 1 – черная.

```
rules = {
      (0, 0, 0): 0,
      (0, 0, 1): 1,
      (0, 1, 0): 1,
      (0, 1, 1): 1,
      (1, 0, 0): 1,
      (1, 0, 1): 0,
      (1, 1, 0): 0,
      (1, 1, 1): 0,
}
```

Собственно напишем функцию

```python
def generation(line):
```

На вход функции подается строчка, которую надо преобразовать в массив целых чисел.
Определим длину массива

```python
    s = np.array([int(i) for i in line])
    size = len(s)
```

Нам нужно десятое поколение клеточного автомата, поэтому вычисления повторятся в цикле соответствующее число раз

```python
    for i in range(10):
```

Учтем условие о том, что поле представляет собой бесконечную полосу клеток (правый сосед последней клетки – это первая клетка, а левый сосед первой клетки – последняя).
Получим состояние клетки и двух её соседей на текущем поколении

```
n3 = np.array((np.roll(s, 1), s, np.roll(s, -1)))
```

Пересчитаем массив на следующем поколении, преобразовав `n3` от i-ой клетки в кортеж (ключ словаря) и найдем элементы словаря

```
s = np.array([rules[tuple(n3[:, i])] for i in range(size)])
```

Выйдя из цикла, вернем результат функции, преобразовав массив s в строку

```
return "".join(str(i) for i in s)
```