Язык Python. Часть 2

Лекция 16

Интерактивность

- Иногда недостаточно просто что-то нарисовать
- Иногда желательна интерактивность
- Есть разные формы интерактивности
- Начнем с простейшей: отслеживание движений мыши и нажатий

Пример

```
# полный код - в репозитории

def on_move(event):
    if event.inaxes:
        print(f'data coords {event.xdata} {event.ydata},',
              f'pixel coords {event.x} {event.y}')


def on_click(event):
    if event.button is MouseButton.LEFT:
        print('disconnecting callback')
        plt.disconnect(binding_id)


binding_id = plt.connect('motion_notify_event', on_move)
```

Разберем пример

- В модуле plt есть функция connect
- Связывает событие с реакцией на него
- Событие определяется строкой
- Из предопределенного набора (важно не опечататься)

Разберем пример

- Действие определяется функцией
- Привязку можно отменить функцией disconnect
- Но надо знать, что отменять
- Каждая привязка имеет свой идентификатор
- Его возвращает connect и принимает disconnect

Развиваем интерактивность

- Естественное развитие - влиять на изображение
- Водим мышкой, что-то нажимаем - и видоизменяем изображение
- Обработчики делать умеем
- Надо изменить логику
- Вместо печати будем воздействовать на Axes

Новый пример

- Нарисуем что-нибудь
- Будем использовать курсор как "прицел"
- От курсора пусть идут вертикальная и горизонтальная линии
- При перемещении перерисовывается
- И текст с координатами мышки

Новый пример

- Заведем вертикальную линию
- Из точек с одной координатой x и разными y
- Аналогично заведем горизонтальную
- Будем ловить событие "движение мыши"

Новый пример

- Будем определять, где мышка
- Для начала - над Axes или нет
- Если над Axes, то в каких координатах
- Нужно нарисовать линии и убрать старые

Новый пример

- Но в явном виде убирать ничего не надо
- Не надо убирать старую линию как элемент Axes и создавать новую
- Можно просто поменять набор точек
- И matplotlib все перерисует в соответствии с новыми данными

Новый пример

- Еще одно удобство - не надо рисовать обычные линии с повторяющимися значениями по координате
- Уже есть методы для вертикальных и горизонтальных линий
- Еще будем держать текст
- И у него тоже можно обновлять содержимое
- Не меняя объекта

Новый пример

- В итоге нужно держать несколько объектов для одной цели
- Это располагает к тому, чтобы завести класс
- И все нужные объекты держать как атрибуты его экземпляра
- Ссылку на метод передадим в качестве обработчика

Пример: начало

```
 1 import matplotlib.pyplot as plt
 2 import numpy as np
 3
 4
 5 class Cursor:
 6     def __init__(self, ax):
 7         self.ax = ax
 8         self.horizontal_line = ax.axhline(color='k', lw=0.8
 9         self.vertical_line = ax.axvline(color='k', lw=0.8,
10         self.text = ax.text(0.72, 0.9, '', transform=ax.tra
11
12     def set_cross_hair_visible(self, visible):
13         need_redraw = self.horizontal_line.get_visible() !=
14         self.horizontal_line.set_visible(visible)
15         self.vertical_line.set_visible(visible)
```

Пример: продолжение

```
def on_mouse_move(self, event):
    if not event.inaxes:
        need_redraw = self.set_cross_hair_visible(False)
        if need_redraw:
            self.ax.figure.canvas.draw()
    else:
        self.set_cross_hair_visible(True)
        x, y = event.xdata, event.ydata
        self.horizontal_line.set_ydata([y])
        self.vertical_line.set_xdata([x])
        self.text.set_text('x=%1.2f, y=%1.2f' % (x, y))
        self.ax.figure.canvas.draw()
```

Пример: окончание

```
 1 x = np.arange(0, 1, 0.01)
 2 y = np.sin(2 * 2 * np.pi * x)
 3
 4 fig, ax = plt.subplots()
 5 ax.set_title('Simple cursor')
 6 ax.plot(x, y, 'o')
 7 cursor = Cursor(ax)
 8 cursor.set_cross_hair_visible(False)
 9 fig.canvas.mpl_connect('motion_notify_event', cursor.on_mou
10
11 plt.show()
```

Анимация

- Иногда нужно показывать не фиксированный момент
- А данные, изменяющиеся в момент процесса
- Или показывать протяженный во времени процесс
- Для этого есть возможность анимации

Общий подход

- Вызываем специальную функцию
- Передаем параметры анимации: интервал между "кадрами" и т.п.
- И функцию
- Функция вызывается с заданной частотой
- Внутри функции делаем нужные изменения

Пример

```
 1 import numpy as np
 2 import matplotlib.pyplot as plt
 3 import matplotlib.animation as animation
 4
 5 fig, ax = plt.subplots()
 6
 7 x = np.arange(0, 2*np.pi, 0.01)
 8 line, = ax.plot(x, np.sin(x))
 9
10
11 def animate(i):
12     line.set_ydata(np.sin(x + i / 50))  # update the data.
13     return line,
14
15
```

Что еще

- Трехмерные изображения
- "Картинки"
- Преобразования
- Стили

Работа с SQL

- SQL - механизм работы с данными
- Универсальный, вне языков программирования
- Язык запросов
- Для создания, изменения и извлечения данных

Мотивация

- Мы уже знаем одну схему работы
- Хранить данные в csv
- Читать в Pandas, вызывать разные методы
- Зачем что-то еще ?

Мотивация

- А если данных много ?
- В память не влезают
- Ну мы можем использовать ленивые конструкции
- Памяти нам хватит, но работать будет долго

Мотивация

- Можно придумать вспомагательные структуры
- Например, если много пользователей
- И много данных о каждом
- Будем хранить в csv

Мотивация

- А рядом - отдельный csv с двумя полями
- id пользователя и позиция (номер байта) в первом файле
- Этот файл влезет в память
- И поможет быстро найти запись о пользователе по id

Мотивация

- Запросы об одних пользователях могут идти чаще, чем о других
- Самые популярные можно хранить в памяти
- И находить их еще быстрее
- Это называется "cache"

Мотивация

- Методов ускорения доступа к данным много
- Но совсем не хочется реализовывать их самостоятельно
- И даже в библиотеке - нежелательно
- Не хотим завязываться на язык программирования
- И так легче реализовывать оптимизации

SQL

- Так родился SQL
- Structured Query Language
- Универсальный язык запросов
- CRUD - create, read, update, delete

А причем здесь Python ?

- Данные, найденнные Python-скриптами, хотим добавлять в SQL-базы
- Чтобы потом читать, обновлять, удалять
- Не обязательно из Python
- Но можно и из Python
- А потом - анализировать, визуализировать и т.п.

Взаимодействие Python и SQL

- Есть два типа взаимодействия: клиент-серверное и библиотечное
- В клиент-серверное есть отдельное приложение - SQL-сервер
- Оно работает отдельно
- Возможно, на другой машинке

Клиент-серверное взаимодействие

- С сервером можно соединиться
- После чего - посылать запросы, получать ответы
- Популярные SQL-серверы: Postgres, MySQL
- Это более "правильная" схема, но сложнее в настройке

Библиотечное взаимодействие

- Обращаемся к библиотеке сразу с запросом
- Библиотека хранит данные на той же машинке
- И управляет ими
- Популярная библиотека: sqlite

sqlite

- Встроена в Python
- import sqlite3
- После импортирования вызываем функцию connect
- Но это не то соединение, которое в клиент-серверном случае

sqlite

- connect в sqlite - значит создание файла
- В котором будут хранится данные
- И который будет подразумеваться в последующих запросах
- Возвращает объект класс Connection

Пример

```
1 import sqlite3
2
3 conn = sqlite3.connect('db')
4 print(conn)
```

Закрытие соединения

- Открытое соединение - ресурс
- Что-то вроде открытого файла
- Надо закрывать
- По возможности - через with/as

Пример

```
1 import sqlite3
2
3 with sqlite3.connect('db') as conn:
4     print(conn)
```

Основные понятия SQL

- Таблица (отношение) - набор строк/записей
- Таблицы имеют имена и структуру
- Структура таблицы - набор описаний полей
- Описание поля - имя и тип данных

Основные понятия SQL

- Основные типы данных: int, text, char()
- int - целое число
- text - строка произвольной длины
- char() - строка фиксированной длины

Пример

```
1 import sqlite3
2
3 with sqlite3.connect('db') as conn:
4     conn.execute("CREATE TABLE movie(title text, year int, s
```

Разберем пример

- Выполняем SQL-запрос на создание таблицы
- CREATE TABLE - начало запроса, обозначение операции
- Количество пробелов неважно
- Важно, чтобы слова были разделены

Разберем пример

- Регистр букв безразличен
- Но принято команды писать заглавными буквами
- А имена - строчными
- Например, movie - имя создаваемой таблицы

Разберем пример

- В скобках - перечень описаний полей
- В нашем случае - три поля
- Описания разделяются запятыми
- В каждом описании - имя поля и тип
- Именно в таком порядке

Разберем пример

- int - целочисленный тип
- text - строка потенциально произвольной длины
- char(3) - строка фиксированной длины
- В данном случае - 3 символа

Зачем разные строковые типы

- Бывают строки заведомо ограниченной длины
- Коды валют, стран, языков и т.п.
- Базе данных удобно с ними работать
- Потому что размер данных фиксирован

Зачем разные строковые типы

- А бывают тексты произвольной длины
- Например, комментарий в форме обратной связи
- База данных будет в каждой записи выделять нужное место
- Работать будет помедленнее, но место использует экономнее

Запуск на сушествующей базе

- При первом запуске база будет создана
- Таблиц там изначально не будет
- Мы своим запросом ее создаем
- При повторном запуске все будет по-другому

Запуск на существующей базе

- Файл есть, база есть, а в ней таблица
- Повторное создание - ошибка
- Можно удалить файл 'db' и начать сначала
- Можно уточнить запрос на создание
- Можно удалить таблицу отдельным запросом

Запуск на сушествующей базе

- В запрос на создание можно добавить 'IF NOT EXISTS'
- Если таблица существует - останется нетронутой
- Запрос на удаление таблицы - 'DROP TABLE movie'
- Тут обратная проблема - не сработает, если таблицы нет

Запуск на сушествующей базе

- Можно уточнить запрос
- Добавить 'IF EXISTS'
- 'DROP TABLE movie IF EXISTS'
- Важно не удалить ничего полезного