Notes

Cheat Sheet

USEFUL LINKS:

https://infostart.ru/1c/articles/1570140/ - Работа с 1С через OData

https://django.fun/tutorials/dokerizaciya-django-s-pomoshyu-postgres-gunicorn-i-nginx/ (https://testdriven.io/blog/dockerizing-django-with-postgres-gunicorn-and-nginx/) (https://fixmypc.ru/post/sozdanie-i-zapusk-konteinera-docker-s-django-postgressql-gunicorn-i-nginx/) - Django, Docker, Gunicorn, Nginx

https://stackoverflow.com/questions/45717835/docker-proxy-pass-to-another-container-nginx-host-not-found-in-upstream - связь между приложениями в контейнерах

https://nuancesprog.ru/p/7481/ - открываем порты в докере

https://datatables.net/examples/api/form.html - Datatables

https://disk.yandex.ru/d/_vLcQHuak0Qjug - MDB5, стили от bootstrap

CMD:

pip freeze -l > requirements.txt — создаем требования (нужные библиотеки для работы программы)
pip install -r requirements.txt — установка пакетов из requirements.txt
wsl hostname -I — адрес WSL (win)
ipconfig — IP адреса (win)
wslconfig /l — все подсистемы
wslconfig /u "wslname" — удалить подсистему
wsl –install -d "wslname" — установить определенную систему WSL
wsl -l -v — посмотреть все wsl
wsl --terminate <WSL_name> — выключить WSL

WSL2:

path: \\wsl$ — путь к локальной папке WSL
$ ip -o address — адрес WSL (deb)
$ ip route — адрес Win (deb)
$ sudo -i — работать через root
$ export DISPLAY=$(ip route | awk '{print $3; exit}'):0 — запустить дисплей в WSL (запуск Xlaunch с параметром "-ac")
$ ssh remote_username@remote_host — подключиться к серверу (remote_host = ssh ip)
scp /home/test.txt root@123.123.123.123:/directory — копировать файл с локального компа на сервер (scp [путь к файлу] [имя пользователя]@[имя сервера/ip-адрес]:[путь к файлу])

Environment:

env — название папки виртуального окружения

python -m venv env (.env - папка скрытая) — создание виртуального окружения
env\Scripts\activate — активация виртуального окружения
env\Scripts\deactivate — деактивация виртуального окружения

Git:

git config --global user.name "Ваше Имя"
git config --global user.email "ВашаПочта@example.com"
git reset имяФайла.расширение — убрать файл
git add . (лучше точно указать файл) — добавление всех файлова
git commit -m "Имя комментария" — создание коммита по добавленным файлам
git push origin master (git push) — пуш коммита в удаленный репозиторий

git branch — в какой ветке нахожусь
git branch "название ветки" — создать ветку
git checkout "название ветки" — переход в ветку
git push --set-upstream origin "название ветки" — добавление ветки в самом репозитории
git push -u origin "название ветки" — добавление ветки в самом репозитории
git push origin master — пуш в определенную ветку
git branch --delete "название ветки" — удаление ветки локально
git push origin --delete "название ветки" — удаление ветки в репозитории
git branch -m <oldname> <newname> — переименовать ветку

git pull origin — загрузить все ветки с удаленного репозитория
git pull origin master — загрузить ветку с удаленного репозитория
замержить ветку Б в ветку А:

- *git checkout branchA* — переключаемся на ветку А
- *git merge branchB* — мержим ветку Б в ветку А

git clone —branch <branchname> <remote-repo-url> — клонировать определенную ветку

Python:

$ time python3 test.py — время выполнения скрипта
время выполнения команды:

```
import time
print('---   ---')
start_time = time.time()
# your code
print('Get smth')
print(f'--- {(time.time() - start_time)} seconds ---')
```

Computer science

Компиляторы и интерпретаторы:

Интерпретатор - построчно читает исходный код программы и выполняет инструкции, содержащиеся в текущей строке, потом переходит к след. строке. Интерпретатор должен присутствовать все время выполнения программы. Интерпретация выполняется при каждом запуске программы

Компилятор - преобразует программу в объектный код, который может напрямую выполняться компьютером.
Компиляция выполняется один раз

API
API (Application Programming Interface) - описание способов, которыми одна программа может взаимодействовать с другой. Набор компонентов с помощью которых программа может взаимодействовать с другой программой.

MVC (Model View Controller)
Model - https://habr.com/ru/post/321050/
View Controller - https://habr.com/ru/post/322700/

Internet

Архитектура сети:
Клиент-сервер - существует хост называемый сервером, который обслуживает запросы от других хостов клиентов. (Youtube и т.д.)
P2P - взаимодействие между хостами без серверов (Torrent, Skype, мессенджеры(частично на сервере))

Уровни интернета:
Прикладной
Транспортный - логическое соединение между процессами (клиентский или серверный процесс
на различных хостах)
Сетевой - логическое соединение между хостами
Канальный
Физический

Виды задержек:
Задержка обработки - проверка заголовков пакета и определение маршрута, проверка ошибок.
Задержка ожидания - очередь, ожидание передачи в линию связи.
Задержка передачи - время проталкивания битов пакета в линию связи (зависит от скорости передачи маршрутизаторов и величины пакета).
Задержка распространения - движения битов по физической среде (провода, радиоволны)(зависит от расстояния и скорости распространения в среде).

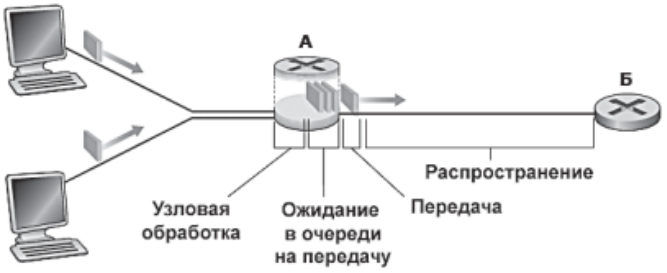


Рис. 1.16. Узловая задержка на маршрутизаторе А

Прикладной уровень:

HTTP:

HTTP - протокол без сохранения состояния (stateless protocol). Относит каждый запрос к независимой транзакции, которая не связана с предыдущим запросом, то есть общение с сервером состоит из независимых пар запрос-ответ.
Пошагово:

1. Клиент инициирует соединение с веб-сервером (отправляет TCP-сегмент)
2. Веб-сервер подтверждает соединение (отправляет ответный TCP-сегмент)
3. Клиент еще раз отправляет сегмент с подтверждением + HTTP запрос
4. Веб-сервер отправляет ответ (запрошенный файл).

Непостоянное HTTP соединение - для каждого объекта (текст, картинка, аудио, видео) устанавливается/разрывается TCP-соединение.
Постоянное HTTP соединение - после запроса объекта оставляет соединение открытым до таймаута.

HTTP REQUEST:

```
GET /somedir/page.html HTTP/1.1      -  строка запроса
Host: www.someschool.edu
Connection: close                    -  строки заголовков
User-agent: Mozilla/5.0
Accept-language: fr
```

Host - адрес хоста на котором размещается объект
Connection - оставлять или разорвать TCP-соединение с сервером
User-agent - наименование браузера пользователя
Accept-language - предпочитаемый язык клиента

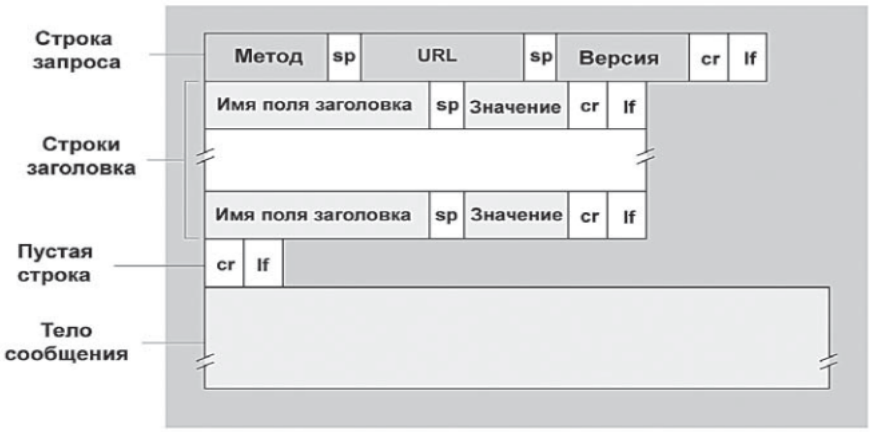


Рис. 2.8. Общий формат сообщения-запроса HTTP

HTTP RESPONSE:

```
HTTP/1.1 200 OK                                          - строка состояния
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT                      - строки заголовков
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(данные данные данные данные данные...)                  - тело сообщения
```

Connection - сервер собирается разорвать TCP-соединение
Date - дата и время создания ответа сервером
Server - наименование веб-сервера
Last-modified - время последнего изменения объекта
Content-length - число байт в пересылаемом объекте
Content-type - тип объекта

Рис. 2.9. Общий формат сообщения-ответа протокола HTTP

HTTPS:

HTTPS (от англ. HyperText Transfer Protocol Secure) - расширение протокола HTTP, поддерживающее шифрование посредством криптографических протоколов SSL и TLS

Для работы с конфиденциальными данными и финансовыми инструментами.
SSL-сертификат подтверждает подлинность сайта, шифрует данные

SOCKET:

Сокет - интерфейс между прикладным и транспортным уровнями внутри хоста (Application Programming Interface, API между приложением и сетью)

POST & PUT:

POST не идемпотентен
Сервер задает имена(id) используем POST
POST /topic/hello?message = Здесь
POST /topic/hello?message = был
POST /topic/hello?message = Вася
== Здесь был Вася

PUT идемпотентен (Метод HTTP является идемпотентным, если повторный идентичный запрос, сделанный один или несколько раз подряд, имеет один и тот же эффект, не изменяющий состояние сервера)
Пользователь задает имена(id) объекту используем PUT
PUT /topic/hello?message = Здесь
PUT /topic/hello?message = был
PUT /topic/hello?message = Вася
== Вася
https://stackoverflow.com/questions/630453/what-is-the-difference-between-post-and-put-in-http

COOKIE:

При первом HTTP-запросе сервер присваивает клиенту идентификационный номер (Set-cookie: 1231). При дальнейших запросах к серверу передается и идентификационный номер клиента, что позволяет не вводить каждый раз одни и те же данные.

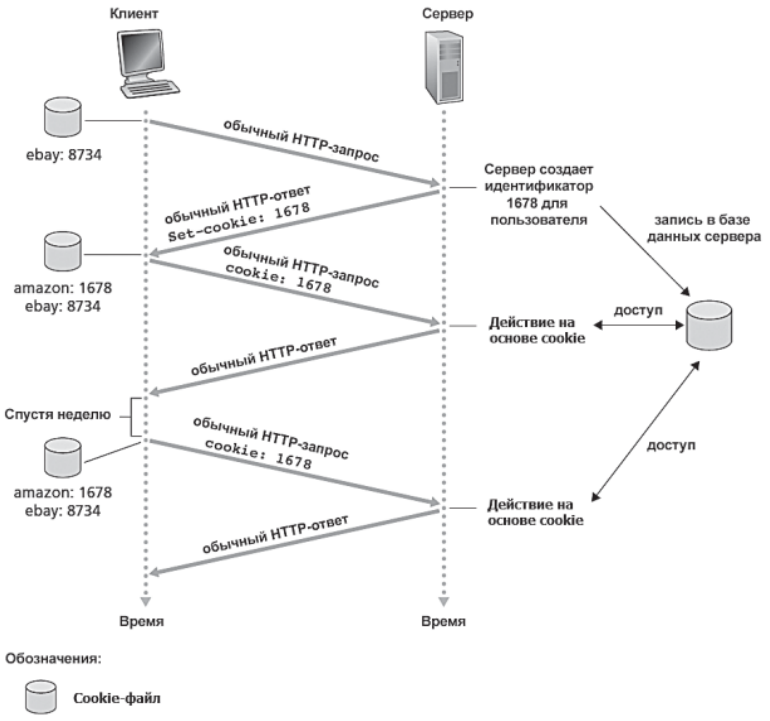


Рис. 2.10. Использование механизма cookie для сохранения состояния пользовательского сеанса

Прокси-сервер (веб-кэш):

Прокси-сервер - это элемент сети, который обрабатывает HTTP-запрос в дополнение к «настоящему» веб-серверу. Для этого на прокси-сервере имеется собственное дисковое хранилище, куда помещаются копии недавно запрошенных объектов. (веб-кэш является и сервером, и клиентом одновременно: когда он получает запросы от браузера и отправляет ему ответы, он выступает в роли сервера, когда он обменивается сообщениями с веб-сервером, то играет роль клиента.)
Чаще всего прокси-серверы применяются для следующих целей:

- обеспечение доступа компьютеров локальной сети к сети Интернет;
- кэширование данных: если часто происходят обращения к одним и тем же внешним ресурсам для снижения нагрузки на канал во внешнюю сеть и ускорения получения клиентом запрошенной информации;
- сжатие данных: прокси-сервер загружает информацию из Интернета и передаёт информацию конечному пользователю в сжатом виде для экономии внешнего сетевого трафика клиента или внутреннего — организации, в которой установлен прокси-сервер;
- защита локальной сети от внешнего доступа: например, можно настроить прокси-сервер так, что локальные компьютеры будут обращаться к внешним ресурсам только через него, а внешние компьютеры не смогут обращаться к локальным вообще (они «видят» только прокси-сервер);
- ограничение доступа из локальной сети к внешней: например, можно запрещать доступ к определённым веб-сайтам, ограничивать использование интернета

каким-то локальным пользователям, устанавливать квоты на трафик или полосу пропускания, фильтровать рекламу и вирусы;

- анонимизация доступа к различным ресурсам: прокси-сервер может скрывать сведения об источнике запроса или пользователе. В таком случае целевой сервер видит лишь информацию о прокси-сервере, например IP-адрес, но не имеет возможности определить истинный источник запроса; существуют также *искажающие прокси-серверы*, которые передают целевому серверу ложную информацию об истинном пользователе;
- обход ограничений доступа: используется, например, пользователями стран, где доступ к некоторым ресурсам ограничен законодательно и фильтруется.

Рис. 2.11. Запросы клиентов через прокси-сервер

GET с условием - прокси-сервер проверяет, изменился ли файл на сервере (отправляет запрос со специальным заголовком). Данный метод просит сервер переслать объект только в том случае, если он был изменен с момента указанного в спец. заголовке.

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 7 Sep 2011 09:23:24
```

FTP:

Протокол с сохранением состояния. File Transfer Protocol
Протокол передачи файлов. Как и HTTP работает поверх TCP. Протокол HTTP отсылает строки заголовков запроса и ответа в одном и том же TCP соединении вместе с передаваемым файлом (внутри полосы). FTP использует два параллельных TCP-соединения (вне полосы):
Управляющее соединение - для отправки контрольной информации между хостами.
Соединение данных - для передачи самого файла.
FTP отправляет ровно один файл через это соединение данных и затем закрывает его. Если в течение этого же сеанса пользователь запрашивает передачу другого файла, то для него FTP открывает еще одно соединение данных. Таким образом, при передаче файлов управляющее соединение остается открытым в течение всего пользовательского сеанса, а для передачи каждого файла внутри этого сеанса создается новое соединение данных (то есть соединение данных в FTP является непостоянным).

SMTP:

Протокол электронной почты. Simple Mail Transfer Protocol
Ограничивает тело любого сообщения почты семиразрядным форматом ASCII (преобразовывает двоичные мультимедийные данные в ASCII перед отправкой и при приеме декодирует обратно)
SMTP не использует каких-либо промежуточных серверов для отправки почты, даже когда два почтовых сервера располагаются на противоположных концах мира.
SMTP использует постоянные соединения: если сервер-источник пытается отправить несколько сообщений одному и тому же серверу-получателю, он может направить их все через одно TCP соединение.

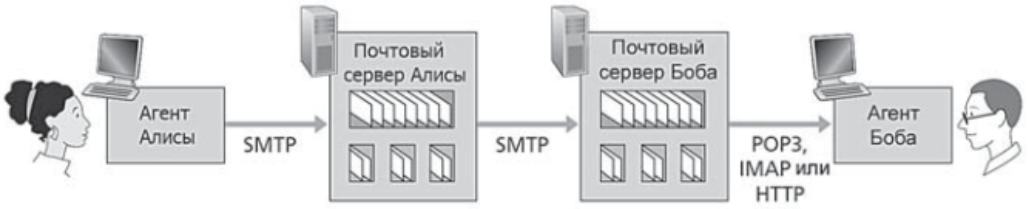


Рис. 2.18. Протоколы электронной почты и их взаимодействующие компоненты

POP3 - чтение и удаление писем
IMAP - чтение, удаление, создание каталогов и папок, возможность получать часть сообщения (например только заголовок)
HTTP - через веб-интерфейс (POP3 и IMAP не используются)

DNS - служба каталогов интернета:

DNS (Domain Name System) - распределенная база данных, реализованная с помощью иерархии DNS-серверов; протокол прикладного уровня, позволяющий хостам обращаться к этой базе данных
Службы DNS:

- трансляция имен хостов в IP-адреса
- назначение псевдонимов хостам
- назначение псевдонимов почтовым серверам
- распределение нагрузки (для одного сайта может быть несколько серверов с разными IP, DNS-сервер выдает весь набор IP-адресов, но в определенном порядке (DNS ротация), так как клиент отправляет запрос первому из списка адресов)

Три класса DNS-серверов в распределенной базе данных:

- *корневые DNS-серверы* (В Интернете существует 13 корневых DNS-серверов (обозначаются латинскими буквами от A до M), также для безопасности за каждым из них есть реплицированные серверы)
- *DNS-серверы верхнего уровня* (отвечают за домены верхнего уровня, такие как com, org, net, edu, а также gov и национальные домены верхнего уровня, такие как uk, fr, ca, jp, ru и т.д.)
- *авторитетные DNS-серверы* (Каждая организация, имеющая публично доступные хосты (веб-серверы или почтовые серверы) в Интернете, должна

предоставить также доступные DNS-записи, которые сопоставляют имена этих хостов с IP-адресами и которые находятся на авторитетном DNS-сервере)

Дополнительный класс, не обязательно принадлежит иерархии серверов:

- *локальный DNS-сервер* (есть у каждого провайдера, запросы сначала идут на локальный DNS-сервер, от него в иерархию DNS-серверов)

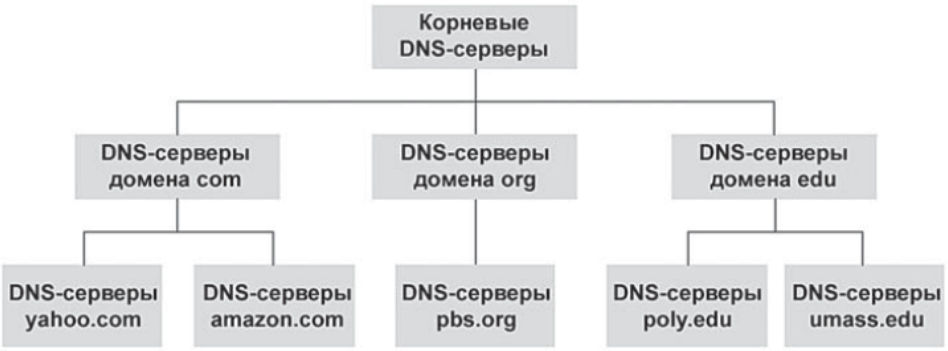


Рис. 2.19. Часть иерархической структуры DNS-серверов

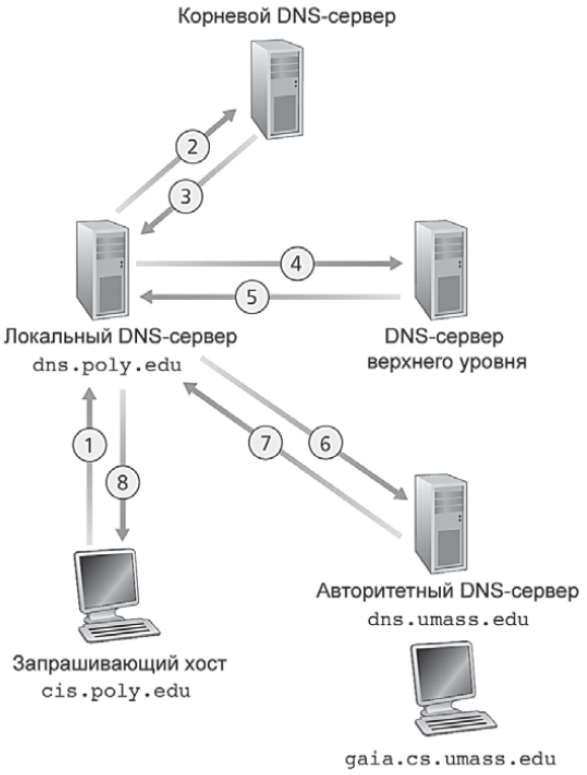


Рис. 2.21. Взаимодействие различных DNS-серверов

Запрос, отправленный с cis.poly.edu к серверу dns.poly.edu, является рекурсивным, так как dns.poly.edu получает информацию по поручению cis.poly.edu. Остальные три запроса являются итеративными, так как все три ответа непосредственно возвращаются к тому, кто запрашивал, а именно к dns.poly.edu.

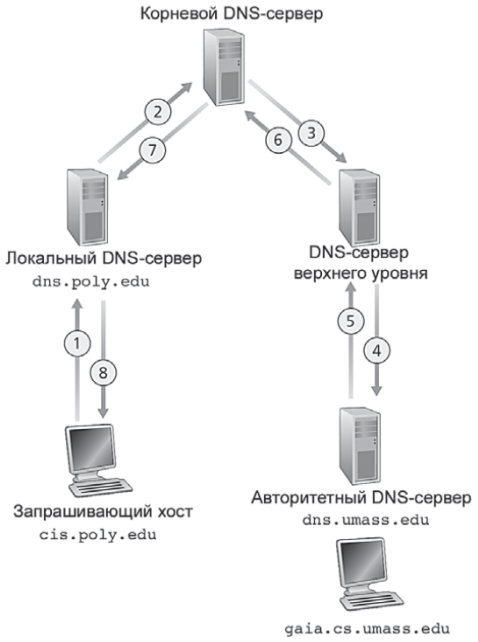


Рис. 2.22. Рекурсивные запросы DNS

DNS-кэширование - сервер сохраняет информацию в локальной памяти (кэше). Так как имена хостов и IP-адреса не обязательно бывают постоянными, DNS-серверы обычно сбрасывают информацию в своем кэше через какой-то период времени (в некоторых случаях это составляет два дня).
Ресурсная запись - запись о соответствии имени и служебной информации DNS-сервера. (Имя, Значение, Тип, Время жизни).
DNS сообщения - два типа: DNS-запрос, DNS-ответ

Рис. 2.23. Формат сообщений DNS

Транспортный уровень:

Мультиплексирование - сбор фрагментов данных, поступающих на транспортный уровень хоста-отправителя из различных сокетов, создание сегментов путем присоединения заголовка к каждому фрагменту и передача сегментов сетевому уровню.

Демультиплексирование - доставка данных сегмента транспортного уровня нужному, соответствующему сокету.

Протокол надежной передачи данных (reliable data transfer, rdt)
Протокол с автоматическим запросом повторной передачи (automatic repeat request, ARQ) - для разрешения проблем искажения битов используются доп. механизмы:

1. обнаружение ошибки (дополнительные специальные биты в поле контрольной суммы)
2. обратная связь с передающей стороной (обратная связь заключается в посылке положительных или отрицательных квитанций, минимальная длина один бит, 0 или 1)
3. повторная передача пакета (пакет, при передаче которого были зафиксированы ошибки, подлежит повторной отправке передающей стороной)

Протокол с ожиданием подтверждений - находясь в состоянии ожидания квитанции, не может принимать новые пакеты от верхнего уровня, прием новых пакетов возможен только после получения положительной квитанции для текущего пакета
Дублирование пакетов - решается присвоением пакету порядкового номера (0 или 1). Также порядковый номер пакета добавляется в квитанцию-ответ
Потеря пакетов - решается ожиданием пакета в течение какого-либо промежутка времени, по окончании которого посчитает пакет потерянным
Передача с конвейеризацией - вместо отправки одного пакета и ожидания ответа, отправитель посылает несколько пакетов без ожидания подтверждений

Механизмы обеспечивающие надежную передачу данных:

- контрольная сумма (обнаружение битовых ошибок в пакете)
- таймер (время ожидания квитанции/пакета, по истечении времени отправляется повторно)
- порядковый номер (нумерация передаваемых пакетов, позволяет обнаружить потерю или дублирование пакета)
- подтверждение (генерируется принимающей стороной и указывает, что соответствующий пакет или группа пакетов получены)
- отрицательное подтверждение (используется получателем для указания некорректности полученного пакета)
- окно, конвейеризация (ограничивают диапазон порядковых номеров для передачи пакетов)

TCP:

Transmission control protocol, TCP - поддерживает передачу с установлением соединения (предварительный обмен управляющей информацией между клиентом и сервером) и надежную передачу данных (гарантированное получение пакетов без ошибок и потерь в строго определенном порядке).
Контроль перегрузки - если участок сети перегружен, заставляет передаваемый процесс снижать нагрузку на сеть.
Когда серверный процесс запущен, клиентский процесс может инициировать TCP-соединение с сервером. При этом клиент указывает адрес входного сокета сервера, а именно IP-адрес серверного хоста и номер порта сокета. После создания своего сокета клиент инициирует тройное рукопожатие и устанавливает

TCP-соединение с сервером. Данное рукопожатие полностью невидимо для клиентской и серверной программ и происходит на транспортном уровне.
Во время тройного рукопожатия клиентский процесс обращается к входному сокету серверного процесса, в ответ на это серверный процесс создает сокет соединения, который предназначен конкретному клиенту. Сокет соединения, предназначенный для клиента организует соединение.

TCP обеспечивает *дуплексную* передачу файлов (при соединение двух хостов, данные могут передаваться в двух направлениях).
TCP является *двухточечным* - соединение устанавливается между единственной парой отправитель-получатель.
Процесс передачи данных от клиентского процесса серверному

1. Тройное рукопожатие
2. *Процесс клиента* передает *данные* через сокет в *буфер передачи* (создается при установлении соединения)
3. TCP составляет пары из фрагментов *данных* клиента и TCP-заголовков, формирую *TCP-сегменты*
4. *Сегменты* передаются на сетевой уровень, где они по отдельности инкапсулируются в *IP-дейтаграммы*
5. *IP-дейтаграммы* передаются в сеть
6. *Сегменты* помещаются в *буфер получения* сервера
7. Приложение считывает *данные* из *буфера получения*

TCP-сегмент:
Порядковый номер - номер первого байта в сегменте в потоке байт (поток данных делится на сегменты в зависимости от MSS, максимального количества данных в сегменте)
Начальный порядковый номер выбирается случайно, чтобы минимизировать вероятность нахождения в сети сегмента, который был сформирован TCP-соединением между этими же двумя хостами

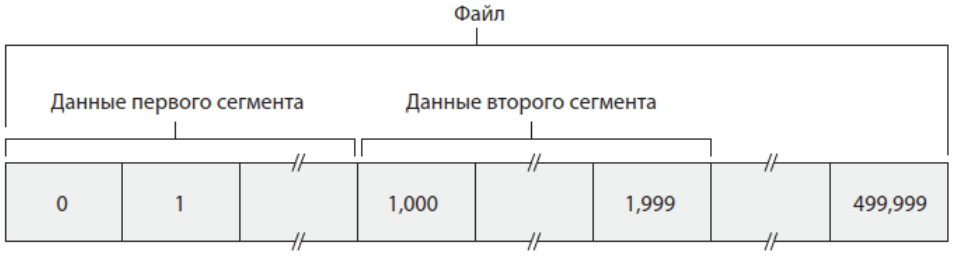


Рис. 3.30. Разделение файла данных на TCP-сегменты

Номер подтверждения хоста А - это порядковый номер следующего байта ожидаемого хостом А от хоста Б. (В качестве примера, предположим, что хост А получил от хоста Б сегмент, содержащий байты с 0 по 535 и сегмент, содержащий байты с 900 по 1000. По какой-то причине хост А все еще не получил байты с 536 по 899. В этом примере, хост А продолжает ждать байт 536 (и далее) в порядке восстановления потока данных хоста Б. Таким образом, следующий сегмент от хоста А к хосту Б содержит значение 536 в поле номера подтверждения. Поскольку TCP квитирует принятые данные до первого отсутствующего байта, говорят, что он поддерживает общее квитирование (рукопожатие))

Рис. 3.29. Структура TCP-сегмента

Выборочное время оборота (RTT) для сегмента - время между отправкой сегмента протоколу IP и до прибытия подтверждения о получении сегмента

Рис. 2.29. Два сокета серверного процесса

UDP:

User datagram protocol, UDP - простая модель передачи, без установления соединения, упорядочивания и целостности данных (используется в чувствительных ко времени приложениях (потоковое видео, аудио, онлайн игры, сервера получающие небольшие запросы от огромного числа клиентов), предпочтительней сбросить пакеты, чем ждать). Нет ограничений по скорости.
Нынешние транспортные протоколы не могут обеспечить гарантированное время доставки и пропускной способности.
Многие брандмауэры настроены на блокирование UDP-трафика, поэтому некоторые потоковые приложения разрабатываются с возможностью использования TCP как резервного.
Отличия протокола UDP (от TCP):

1. Более полный и точный контроль приложения за процессом передачи данных (просто упаковывает данные и сразу отправляет на сетевой уровень, нет лишних механизмов как в TCP);
2. Отсутствует установление соединения;
3. Не заботиться о состоянии соединения (не хранит информацию связанную с дополнительными механизмами, поэтому способен поддерживать намного больше клиентов)
4. Небольшой заголовок пакетов (8 байт против 20 байт у TCP)

UDP сегмент - заголовок состоит из четырех полей по 2 байта каждый.
Контрольная сумма применяется для определения, произошло ли искажение битов UDP сегмента в ходе перемещения.

Рис. 3.7. Структура UDP-сегмента

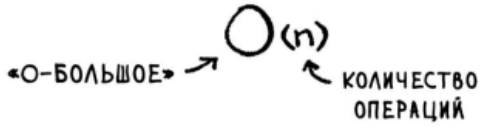
Приложение	Протокол прикладного уровня	Нижерасположенный транспортный протокол
Электронная почта	SMTP	TCP
Удаленный терминальный доступ	Telnet	TCP
Всемирная паутина	HTTP	TCP
Передача файлов	FTP	TCP
Удаленный файловый сервер	NFS	Обычно UDP
Потоковый мультимедийный контент	Обычно проприетарный	UDP или TCP
Интернет-телефония	Обычно проприетарный	UDP или TCP
Сетевое управление	SNMP	Обычно TCP
Протокол маршрутизации	RIP	Обычно TCP
Трансляция имен	DNS	Обычно TCP

Рис. 3.6. Популярные Интернет-приложения и используемые ими протоколы транспортного уровня

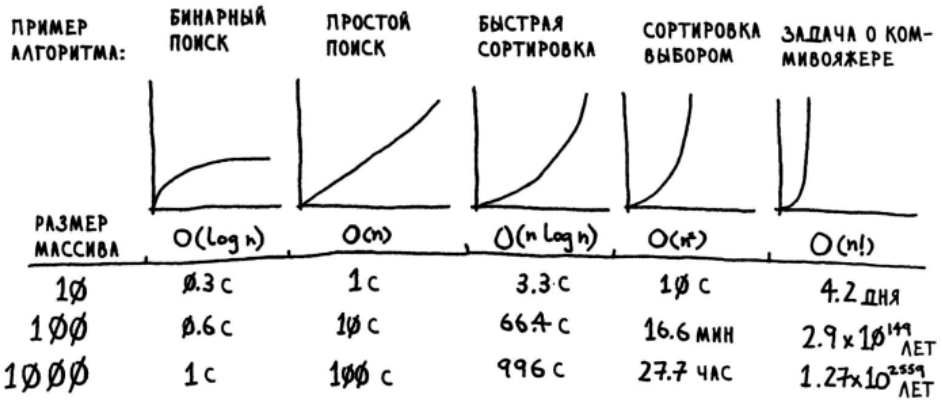
Алгоритмы

О-большое

О-большое - описывает скорость работы алгоритма. (скорость работы измеряется ростом количества операций)
Константа не важна при разной скорости работы алгоритмов, но может быть важна при одинаковых по скорости работы алгоритмах, например быстрая сортировка и сортировка слиянием (константа у сортировки слиянием больше)



Как записывается «О-большое»

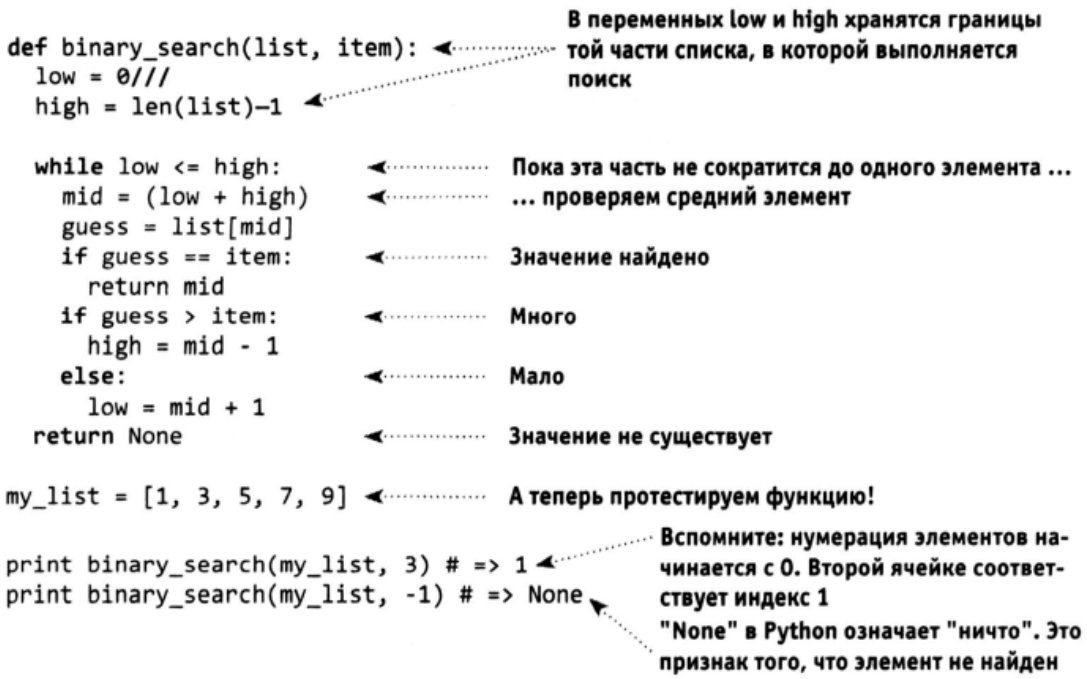


Бинарный поиск

Скорость работы - O(logn)
Работает, если список отсортирован

Двоичный (бинарный) поиск (также известен как метод деления пополам или дихотомия) — классический алгоритм поиска элемента в отсортированном массиве (векторе), использующий дробление массива на половины.

1. Определение значения элемента в середине структуры данных. Полученное значение сравнивается с ключом.
2. Если ключ меньше значения середины, то поиск осуществляется в первой половине элементов, иначе — во второй.
3. Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.
4. Процесс продолжается до тех пор, пока не будет найден элемент со значением ключа или не станет пустым интервал для поиска.



Быстрая сортировка

Скорость работы (средний случай) - O(nlogn)
Худший случай - O(n)
Лучший случай - O(nlogn)
Зависит от выбора опорного элемента
Общая идея алгоритма состоит в следующем:

- Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.
- Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
- Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

```
def quicksort(array):
  if len(array) < 2:
    return array          <..... Базовый случай: массивы с 0 и 1 элементом уже "отсортированы"
  else:
    pivot = array[0]      <..... Рекурсивный случай
    less = [i for i in array[1:] if i <= pivot]   <..... Подмассив всех элементов, меньших опорного

    greater = [i for i in array[1:] if i > pivot] <..... Подмассив всех элементов, больших опорного

    return quicksort(less) + [pivot] + quicksort(greater)

print quicksort([10, 5, 2, 3])
```

Сортировка слиянием

Скорость работы (средний случай) - O(nlogn)

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.

Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по возрастанию подмассива. Тогда:

3.2. Слияние двух подмассивов в третий результирующий массив.

На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.

3.3. «Прицепление» остатка.

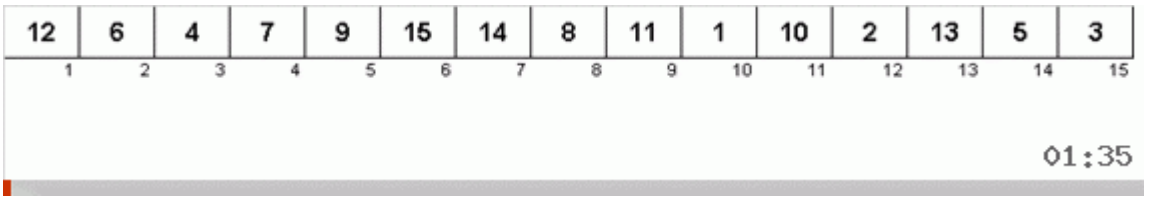
Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

6 5 3 1 8 7 2 4

Сортировка выбором

Скорость работы - $O(n^2)$

Проходим по массиву в поисках максимального элемента. Найденный максимум меняем местами с последним элементом. Неотсортированная часть массива уменьшилась на один элемент (не включает последний элемент, куда мы переставили найденный максимум). К этой неотсортированной части применяем те же действия — находим максимум и ставим его на последнее место в неотсортированной части массива. И так продолжаем до тех пор, пока неотсортированная часть массива не уменьшится до одного элемента.




```
def find Smallest(a r r):
  smallest = arr[0]        <····  Для хранения наименьшего значения
  smallest_index = 0       <····  Для хранения индекса наименьшего значения
  for i in range(1, len(arr)):
    if arr[i] < smallest:
      smallest = arr[i]
      smallest_index = i
  return smallest_index
```

Теперь на основе этой функции можно написать функцию сортировки выбором:

```
def selectionSort(arr):      <····  Сортирует массив
  newArr = []
  for i in range(len(arr)):
      smallest = indSmallest(arr)  <····  Находит наименьший элемент в массиве
      newArr.append(arr.pop(smallest))    и добавляет его в новый массив
  return newArr

print selectionSort([5, 3, 6, 2, 10])
```

Пузырьковая сортировка

Скорость работы - O(n²)

Пузырьковая сортировка - делает по списку несколько проходов. Она сравнивает стоящие рядом элементы и меняет местами те из них, что находятся в неправильном порядке. Каждый проход по списку помещает следующее наибольшее значение на его правильную позицию. В сущности, каждый элемент “пузырьком” всплывает на своё место.

Python

Хэш-таблицы

Другие названия: ассоциативный массив, словарь, хеш
Хеш-функция - функция, получающая данные на вход и возвращающая число
Требования к хеш-функции:

- последовательность (одинаковые входные данные соответствуют одинаковым выходным данным)
- разным выходным данным соответствуют разные выходные данные

Коллизии - двум ключам назначается один элемент массива
Коэффициент заполнения - отношение количества элементов и размера массива хеш-таблицы.

Типы данных:

int & float - числа целые и дробные: 100, 100.23
'str' - строка, неизменяемый упорядоченный: “some string”
[list] - список, **изменяемый** упорядоченный: [1, 2, 4, “text1”, “text2”]
{dict} - словарь, **изменяемый** упорядоченный: {1: “one”, 2: ”two”, 3: ”three”}
до Python 3.7 были неупорядоченным
(tuple) - кортеж, неизменяемый упорядоченный: (“text1”, “text2”, 3, 4)
{set} - множество (только уникальные неизменяемые типы данных), **изменяемый** неупорядоченный: {1, 2, “text1”, “text2”, 7}
Bool - булевы значения, истина и ложь: True (ненулевое число, непустая строка, непустой объект), False (0, None, пустая строка, пустой объект)

Категории типов данных:

- Примитивные типы — они варьируются в зависимости от языка, но самые основные — это целые, числа с плавающей запятой, булевы величины и символы.
- Сложные типы — они состоят из несколько примитивных типов, например, массив или запись (но не хэш). Все сложные типы считаются структурами данных.
- Абстрактные типы — типы, у которых нет конкретной реализации, такие как хэш, множество, очередь и стек.

- Прочие типы — например, указатели (тип, в значении которого хранится ссылка на другое место в памяти).

Динамическая и статическая типизация:

Язык обладает *статической типизацией,* если тип переменной известен во время компиляции, а не выполнения. Типичными примерами таких языков являются Ada, C, C++, C#, JADE, Java, Fortran, Haskell, ML, Pascal, и Scala.
Динамическая типизация — это процесс подтверждения типобезопасности программы во время ее выполнения. Типичными примерами динамически типизированных языков являются Groovy, JavaScript, Lisp, Lua, Objective-C, PHP, Prolog, Python, Ruby, Smalltalk и Tcl.

Python - это динамический язык со строгой типизацией (язык, в котором переменные привязаны к конкретным типам данных, и который выдаст ошибку типизации в случае несовпадения ожидаемого и фактического типов — когда бы не проводилась проверка. Проще всего представить сильно типизированный язык как язык с высокой типобезопасностью)
Ошибка!:

```
x = 1 + "2"
```

List & Tuple:

В *списках* элементы располагаются в произвольных ячейках памяти, но каждый элемент указывает на последующий; списки легко расширяются и справляются со вставкой. Чтение O(n), вставка O(1)
В *кортежах* под количество элементов выделяется конкретное количество ячеек памяти; нельзя расширить кортеж, можно создать новый побольше; мы заранее знаем адрес(индекс) каждого элемента. Чтение O(1), вставка O(n)

Интерполяция строк:

1. %s - `print('Hello, %s' % name)`
2. str.format - `print('Hello, {}'.format(name))`
3. f-строки - `print(f'Hello, {name}!')`
4. Template строки - не позволяют форматировать спецификаторы, более безопасно (https://python-scripts.com/string-formatting#template-strings)

```
from string import Template
t = Template('Hey, $name!')
print(t.substitute(name=name))
```

Разница между is и ==:

is - идентичны ли элементы, ссылаются ли на один и тот же объект в памяти
== - эквиваленты ли объекты, одинаковые, но ссылаются на разные объекты в памяти

Декоратор:

Это функция, которая позволяет обернуть другую функцию для расширения ее функциональности без непосредственного изменения ее кода.

```
def makebold(fn):
```

```
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

def makeitalic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped

@makebold
@makeitalic
def hello():
    return "hello habr"

print hello() ## выведет <b><i>hello habr</i></b>
```

Декораторы могут быть использованы для расширения возможностей функций из сторонних библиотек (код которых мы не можем изменять), или для упрощения отладки (мы не хотим изменять код, который ещё не устоялся).
Также полезно использовать декораторы для расширения различных функций одним и тем же кодом, без повторного его переписывания каждый раз.

Методы класса:

Метод экземпляра - относится к определенному экземпляру (параметр self)
Метод класса - относится к классу, можно изменить сам класс (параметр cls)
Статический метод - является автономным, не может изменять состояние класса или экземпляра, используется обычно для служебных функций (@staticmethod)

Разница между func & func():

func - представляющий функцию объект, можно назначить переменной или передать другой функции.
func() - вызов функции и возвращение результата.

Функция map:

Для применения функции к каждому элементу итерируемого объекта (эффективней цикла for, тк написана на C и оптимизирована + потребляет память только на один элемент по запросу). Возвращает объект типа map, надо отдельно преобразовывать в list()

```
map(function, iterable, [iterable 2, iterable 3, ...])

# Return double of n
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))
Output:
[2, 4, 6, 8]
```

Функция reduce:

Принимает функцию и последовательность - проходит по последовательности. На каждой итерации передаются текущий элемент и выходные данные предыдущего.

```
from functools import reduce
def add_three(x,y):
    return x + y
li = [1,2,3,5]
reduce(add_three, li)
#=> 11
```

Функция filter:

Каждый элемент передается функции, которая включает его в последовательность, если по условию получает True, и отбрасывает в случае False.

```
def add_three(x):
    if x % 2 == 0:
        return True
    else:
        return False

li = [1,2,3,4,5,6,7,8]

[i for i in filter(add_three, li)]
#=> [2, 4, 6, 8]
```

Перевернуть список:

list.reverse() - изменяет список
reversed(list) - возвращает итератор перевернутых элементов
list[::-1] - создает новый список

Объединить списки:

append()
list + list
list comprehension
list.extend()

Copy & deepcopy:

При *поверхностном копировании* создается отдельный объект и ссылки на дочерние элементы (ссылки на их адреса памяти)
При *глубоком копировании* создается новый объект со своим уникальным адресом памяти. Новый объект не ссылается на оригинал

List & array:

List - разные типы данных, арифметические действия добавляют или удаляют элементы из списка, печать без цикла
Array - однородные данные, арифметические действия соответствуют функциям линейной алгебры, используют меньше памяти, для печати нужен цикл

Изменяемые (mutable) и неизменяемые объекты (immutable):

Изменяемые - списки (list), множества (set), байтовые массивы (byte arrays) и словари (dict).
Неизменяемые - целые числа (int), числа с плавающей запятой (float), булевы значения (bool), строки (str), кортежи (tuple), неизменяемое множество (frozen set)

Модуль pickle:

Для сериализации и десериализации (преобразование данных в набор байтов, который обычно сохраняют в файл или передают по сети)

Any() & all():

Any возвращает true, если хоть один элемент в последовательности соответствует условию, то есть является true.
All возвращает true только в том случае, если условию соответствуют все элементы в последовательности.

List comprehension:

Представление списков - позволяет создавать списки в одной строке.
Не надо создавать пустой список, как с циклом for.
Возвращает список, в отличии от map().
Можно использовать условную логику (if) для фильтрации или изменения значения элемента.
При помощи вложенных друг в друга list comprehension можно создавать матрицы

```
new_list = [expression for member in iterable]
>>> squares = [i * i for i in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

С условным выражением:
В условие можно передать функцию

```
new_list = [expression for member in iterable (if conditional)]
>>> sentence = 'the rocket came back from mars'
>>> vowels = [i for i in sentence if i in 'aeiou']
>>> vowels
['e', 'o', 'e', 'a', 'e', 'a', 'o', 'a']
```

Выбор из нескольких вариантов:
В условие также можно передать функцию

```
>>> original_prices = [1.25, -9.45, 10.22, 3.78, -5.92, 1.16]
>>> prices = [i if i > 0 else 0 for i in original_prices]
>>> prices
[1.25, 0, 10.22, 3.78, 0, 1.16]
```

Dict comprehension:

Создаем список букв

```
import string
list(string.ascii_lowercase)
alphabet = list(string.ascii_lowercase)
```

Генерация словаря

```
d = {val:idx for idx,val in enumerate(alphabet)}
```

Тернарный оператор:

Тернарный оператор - конструкция, которая по своему действию аналогична конструкции if-else, но при этом является выражением. Тернарный оператор — единственный в своем роде оператор, требующий три операнда:

```
def abs(number):
    return number if number >= 0 else -number
```

Дескрипторы:

Почитать еще!

ORM:

ORM - технология программирования, прослойка, которая связывает базы данных с концепциями ООП языков программирования.

ORM - это уровень абстракции данных, хранящихся в базе данных с таблицами, строками и столбцами. Он позволяет работать с базами данных, используя знакомые объектно-ориентированные метафоры, которые хорошо работают с кодом. Классы отображаются на таблицы базы данных, атрибуты отображаются на столбцы, и отдельный экземпляр класса представляет строку данных в базе данных.

lambda function:

>>> f = lambda arguments: expression

*>>> f = lambda x: x * x*

Используются как одноразовые анонимные функции.

Несколько аргументов:

>>> f = lambda x, y: x * y

>>> f(5, 2)

10

lambda функция представляет собой одно выражение

Не может включать операторы return, pass, assert или raise в свое тело.

Нет аннотации типов

Можно немедленно вызывать функцию:

>>> (lambda x: x * x)(3)

Функции lambda не имеют названий. При ошибке lambda функции:

>>> File "<stdin>", line 1, in <lambda>

При ошибке обычной функции:

>>> File "<stdin>", line 1, in div_zero

ООП:

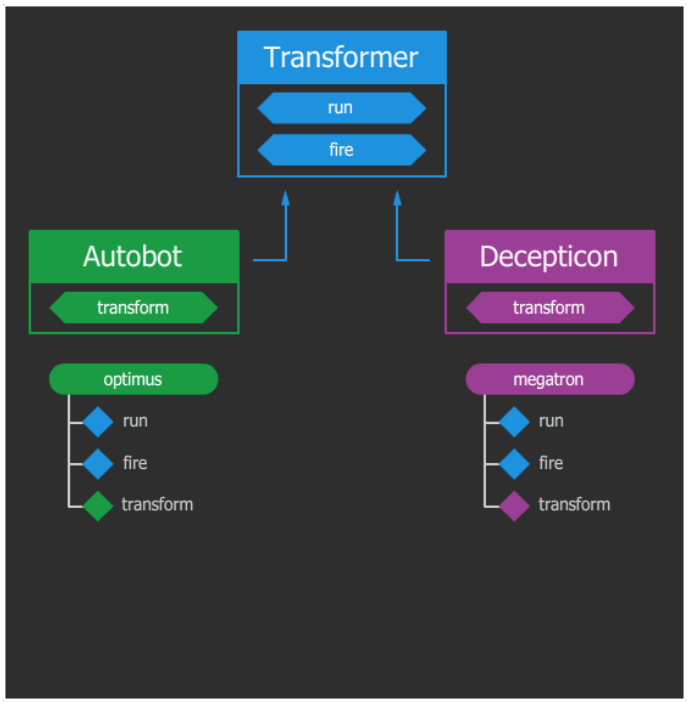
*ООП - п*опытка связать поведение сущности с ее данными и спроецировать объекты реального мира и бизнес-процессов в программный код. Задумывалось, что такой код проще читать и понимать человеком, т. к. людям свойственно воспринимать окружающий мир как множество взаимодействующих между собой объектов, поддающихся определенной классификации.

Ассоциация - традиционно в полях объекта могут храниться не только обычные переменные стандартных типов, но и другие объекты. А эти объекты могут в свою

очередь хранить какие-то другие объекты и так далее, образуя дерево (иногда граф) объектов:

1. *Композиция* - жизненный цикл дочернего объекта совпадает с жизненным циклом родительского.
2. *Агрегация* - жизненный цикл дочернего объекта не зависит от жизненного цикла родительского, и может использоваться другими объектами.

НАСЛЕДОВАНИЕ- это механизм системы, который позволяет наследовать одними классами свойства и поведение других классов для дальнейшего расширения или модификации.



Как при описании отношений двух сущностей определить, когда уместно наследование, а когда — композиция? Можно воспользоваться популярной шпаргалкой: спросите себя, сущность А является сущностью Б? Если да, то скорее всего, тут подойдет наследование. Если же сущность А является частью сущности Б, то наш выбор — композиция.

Абстрактный класс - который содержит один и более абстрактных методов. Абстрактным называется объявленный, но не реализованный метод. Абстрактные классы не могут быть инстанциированы, от них нужно унаследовать, реализовать все их абстрактные методы и только тогда можно создать экземпляр такого класса. В Python отсутствует встроенная поддержка абстрактных классов, для этой цели используется модуль abc (Abstract Base Class)

```
from abc import ABC, abstractmethod

class ChessPiece(ABC):
    # общий метод, который будут использовать все наследники этого
класса
    def draw(self):
        print("Drew a chess piece")
```

```
    # абстрактный метод, который будет необходимо переопределять для
каждого подкласса
    @abstractmethod
    def move(self):
        pass

class Queen(ChessPiece):
    def move(self):
        print("Moved Queen to e2e4")

# Мы можем создать экземпляр класса
q = Queen()
# И нам доступны все методы класса
q.draw()
q.move()
```

ПОЛИМОРФИЗМ- разное поведение одного и того же метода в разных классах.
Полиморфизм - это возможность обработки разных типов данных, т. е. принадлежащих к разным классам, с помощью "одной и той же" функции, или метода. На самом деле одинаковым является только имя метода, его исходный код зависит от класса.
Полиморфизм дает возможность реализовывать так называемые единые интерфейсы для объектов различных классов. Например, разные классы могут предусматривать различный способ вывода той или иной информации объектов. Однако одинаковое название метода вывода позволит не запутать программу, сделать код более ясным.

ИНКАПСУЛЯЦИЯ - ограничение доступа к составляющим объект компонентам (методам и переменным). Инкапсуляция делает некоторые из компонент доступными только внутри класса.
Обобщая можно сказать что данные объекта могут быть:

- публичный (public, нет особого синтаксиса, publicBanana); к атрибуту может получить доступ любой желающий
- защищенный (protected, одно нижнее подчеркивание в начале названия, _protectedBanana); к атрибуту могут обращаться только методы данного класса
- приватный (private, два нижних подчеркивания в начала названия, __privateBanana). то же, что и protected, только доступ получают и наследники класса в том числе
- «магические методы» («magic methods») или «специальные методы» («special methods») например __init__ (инициализатор) является наиболее часто используемым из них и запускается при создании нового объекта класса.

Утиная типизация заключается в том, что вместо проверки типа чего-либо в Python мы склонны проверять, какое поведение оно поддерживает, зачастую пытаясь использовать это поведение и перехватывая исключение, если оно не работает.

АБСТРАКЦИЯ - это выделение основных, наиболее значимых характеристик объекта и игнорирование второстепенных.

Flask

pass

SQL

Структура:

```
SELECT ('столбцы или * для выбора всех столбцов; обязательно')
FROM ('таблица; обязательно')
WHERE ('условие/фильтрация, например, city = 'Moscow'; необязательно')
GROUP BY ('столбец, по которому хотим сгруппировать данные;
необязательно')
HAVING ('условие/фильтрация на уровне сгруппированных данных;
необязательно')
ORDER BY ('столбец, по которому хотим отсортировать вывод;
необязательно')
```

SELECT порядок выполнения:

Порядок обработки предложений:

1. *FROM*
2. *WHERE*
3. *GROUP BY*
4. *HAVING*
5. *SELECT*
6. *ORDER BY*

Команды:

Стандартными командами для взаимодействия с РБД являются `CREATE`, `SELECT`, `INSERT`, `UPDATE`, `DELETE` и `DROP`. Эти команды могут быть классифицированы следующим образом:

- `DDL` — язык определения данных (Data Definition Language)

1	CREATE	Создает новую таблицу, представление таблицы или другой объект в БД
2	ALTER	Модифицирует существующий в БД объект, такой как таблица
3	DROP	Удаляет существующую таблицу, представление таблицы или другой объект в БД

- `DML` — язык изменения данных (Data Manipulation Language)
  CRUD (**Create, Read, Update, Delete**)

1	SELECT	Извлекает записи из одной или нескольких таблиц
2	INSERT	Создает записи

3	UPDATE	Модифицирует записи
4	DELETE	Удаляет записи

- `DCL` — язык управления данными (Data Control Language)

1	GRANT	Наделяет пользователя правами
2	REVOKE	Отменяет права пользователя
3	SET ROLE	Разрешает/запрещает роли для текущего сеанса

- `TCL`— язык управления транзакциями (Transaction Control Language)

1	COMMIT	Подтверждает транзакцию и делает постоянными изменения
2	ROLLBACK	Откат транзакции, отменяет изменения

Ограничения:

Ограничения (constraints) — это правила, применяемые к данным. Они используются для ограничения типа данных, которые могут быть записаны в таблицу. Это обеспечивает точность и достоверность данных в БД.
Ограничения могут применяться либо на уровне столбцов, либо на уровне таблицы.

- `NOT NULL` — колонка не может иметь NULL значение
- `DEFAULT` — значение колонки по умолчанию
- `UNIQUE` — все значения колонки должны быть уникальными
- `PRIMARY KEY` — первичный или основной ключ, уникальный идентификатор записи в текущей таблице
- `FOREIGN KEY` — внешний ключ, уникальный идентификатор записи в другой таблице (таблице, связанной с текущей)
- `CHECK` — все значения в колонке должны удовлетворять определенному условию
- `INDEX` — быстрая запись и извлечение данных

Целостность в БД:

Целостность БД - согласованность (непротиворечивость) данных.
Целостность объекта - для обеспечения целостности объекта (сущности) используется PRIMARY KEY (однозначная идентификации той или иной записи в таблице). Для таблицы может быть создано только одно такое ограничение. Не может быть NULL значения

Целостность домена - для обеспечения целостности домена (предметной области) используется CHECK (гарантирует наличие в определенном столбце только допустимых значений, ограничивая тип, формат или диапазон.).
Целостность ссылок - ограничение базы данных, гарантирующее, что ссылки между данными являются действительно правомерными и неповрежденными.
Обеспечивается использованием FOREIGN KEY (защищает от действий, которые могут нарушить связи между таблицами).
NO ACTION - запрещающий механизм
CASCADE - каскадный механизм
Целостность, определяемая пользователем - позволяет задавать некоторые бизнес-правила, не попадающие ни в какую другую категорию целостности.

Нормализация БД:

https://info-comp.ru/database-normalization
Декомпозиция - вертикальная (разбиение таблицы на несколько таблиц по разным колонкам), горизонтальная (разбиение таблицы на несколько таблиц по определенному признаку (текущие данные и исторические))
В реальности используется нормализация до 3 уровня (редко до 4), так на следующих уровнях падает производительность БД (становится много таблиц).
Нормализация - это процесс удаления избыточных данных.
Нормализация - это метод проектирования базы данных, который позволяет привести базу данных к минимальной избыточности.
Нормализация нужна для устранения аномалий, повышения производительности, повышения удобства управления данными.
Избыточность данных - одни и те же данные хранятся в базе в нескольких местах, именно это приводит к аномалиям.
Процесс нормализации - это последовательный процесс приведения базы данных к эталонному виду, т.е. переход от одной нормальной формы к следующей.
Нормальные формы БД:

- ненормализованная форма или нулевая нормальная форма (UNF) - база не является реляционной (порядок строк и столбцов не имеет значения)
- первая нормальная форма (1NF) - не должно быть дублирующих строк, в каждой ячейке хранится атомарное (одно не составное значение) значение,в столбце хранятся данные одного типа, отсутствуют массивы и списки в любом виде.
- вторая нормальная форма (2NF) - таблица должна иметь ключ, все неключевые столбцы таблицы должны зависеть от полного ключа (если он составной)
- третья нормальная форма (3NF) - отсутствие транзитивной зависимости (неключевые столбцы зависят от значений других неключевых столбцов)
- нормальная форма Бойса-Кодда (BCNF) или усиленная третья нормальная форма (для таблиц с составным ключом) - ключевые атрибуты составного ключа не должны зависеть от неключевых атрибутов.
- четвертая нормальная форма (4NF) - отсутствие нетривиальных многозначных зависимостей (две разные колонки связаны с одной, но не связаны между собой)
- пятая нормальная форма (5NF) - каждая нетривиальная зависимость соединения определяется потенциальным ключом этого отношения.

- доменно-ключевая нормальная форма (DKNF) - каждое наложенное ограничение на таблицу является логическим следствием ограничений доменов и ограничений ключей, которые накладываются на данную таблицу
- шестая нормальная форма (6NF) - относится к хронологическим БД (может хранить не только текущие данные, но и исторические данные, т.е. данные, относящиеся к прошлым периодам времени. Однако такая база может хранить и данные, относящиеся к будущим периодам времени.) - таблица должна удовлетворять всем нетривиальным зависимостям соединения (не может быть подвергнута дальнейшей декомпозиции без потерь).

База данных считается нормализованной, если она находится как минимум в третьей нормальной форме (3NF).

Типы данных:

Точные числовые:

Тип данных	От	До
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Приблизительные числовые:

Тип данных	От	До
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

Дата и время:

Тип данных	От	До
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079

date	Дата сохраняется в виде June 30, 1991	
time	Время сохраняется в виде 12:30 P.M.	
datetime2	YYYY-MM-DD hh:mm:ss[.fractional seconds]	
datetimeoffset	YYYY-MM-DD hh:mm:ss[.nnnnnnn] [{+|-}hh:mm]	

Строковые символьные:

N	Тип данных	Описание
1	char	Строка длиной до 8,000 символов (не-юникод символы, фиксированной длины)
2	varchar	Строка длиной до 8,000 символов (не-юникод символы, переменной длины)
3	text	Не-юникод данные переменной длины, длиной до 2,147,483,647 символов

Строковые символьные (юникод):

N	Тип данных	Описание
1	nchar	Строка длиной до 4,000 символов (юникод символы, фиксированной длины)
2	nvarchar	Строка длиной до 4,000 символов (юникод символы, переменной длины)
3	ntext	Юникод данные переменной длины, длиной до 1,073,741,823 символов

Бинарные:

N	Тип данных	Описание
1	binary	Данные размером до 8,000 байт (фиксированной длины)
2	varbinary	Данные размером до 8,000 байт (переменной длины)
3	image	Данные размером до 2,147,483,647 байт (переменной длины)

Смешанные:

N	Тип данных	Описание
1	timestamp	Уникальные числа, обновляющиеся при каждом изменении строки

2	uniqueidentifier	Глобально-уникальный идентификатор (GUID)
3	cursor	Объект курсора
4	table	Промежуточный результат, предназначенный для дальнейшей обработки

Операторы:

Оператор (operators) — это ключевое слово или символ, которые, в основном, используются в инструкциях `WHERE` для выполнения каких-либо операций. Они используются как для определения условий, так и для объединения нескольких условий в инструкции.

(a = 10, b = 20)

Арифметические:

Оператор	Описание	Пример
+ (сложение)	Сложение значений	a + b = 30
— (вычитание)	Вычитание правого операнда из левого	b — a = 10
* (умножение)	Умножение значений	a * b = 200
/ (деление)	Деление левого операнда на правый	b / a = 2
% (деление с остатком/по модулю)	Деление левого операнда на правый с остатком (возвращается остаток)	b % a = 0

Операторы сравнения:

Оператор	Описание	Пример
=	Определяет равенство значений	a = b -> false
!=	Определяет НЕравенство значений	a != b -> true
<>	Определяет НЕравенство значений	a <> b -> true
>	Значение левого операнда больше значения правого операнда?	a > b -> false
<	Значение левого операнда меньше значения правого операнда?	a < b -> true

>=	Значение левого операнда больше или равно значению правого операнда?	a >= b -> false
<=	Значение левого операнда меньше или равно значению правого операнда?	a <= b -> true
!<	Значение левого операнда НЕ меньше значения правого операнда?	a !< b -> false
!>	Значение левого операнда НЕ больше значения правого операнда?	a !> b -> true

Логические операторы:

N	Оператор	Описание
1	ALL	Сравнивает все значения
2	AND	Объединяет условия (все условия должны совпадать)
3	ANY	Сравнивает одно значение с другим, если последнее совпадает с условием
4	BETWEEN	Проверяет вхождение значения в диапазон от минимального до максимального
5	EXISTS	Определяет наличие строки, соответствующей определенному критерию
6	IN	Выполняет поиск значения в списке значений
7	LIKE	Сравнивает значение с похожими с помощью операторов подстановки
8	NOT	Инвертирует (меняет на противоположное) смысл других логических операторов, например, NOT EXISTS, NOT IN и т.д.
9	OR	Комбинирует условия (одно из условий должно совпадать)
10	IS NULL	Определяет, является ли значение NULL
11	UNIQUE	Определяет уникальность строки

Выражения:

Выражение (expression) — это комбинация значений, операторов и функций для оценки (вычисления) значения. Выражения похожи на формулы, написанные на языке запросов. Они могут использоваться для извлечения из БД определенного набора данных.

ROLLUP

pass

LIKE & REGEX:

LIKE используется для сравнения значений с помощью операторов с подстановочными знаками. Существует два вида таких операторов:

`%` - любое количество символов

`_` - один символ

REGEX позволяет определять регулярное выражение, которому должна соответствовать запись.

`^` - начало строки

`$` - конец строки

`.` - любой символ

[`символы`] - любой из указанных символов

[`начало-конец`] - любой символ из диапазона

`|` - разделяет шаблоны

TOP & LIMIT & ROWNUM:

Данные предложения позволяют извлекать указанное количество или процент записей с начала таблицы. Разные СУБД поддерживают разные предложения.

ORDER BY & GROUP BY:

ORDER BY - сортировка данных по возрастанию (ASC) или по убыванию (DESC).
GROUP BY - группировка данных

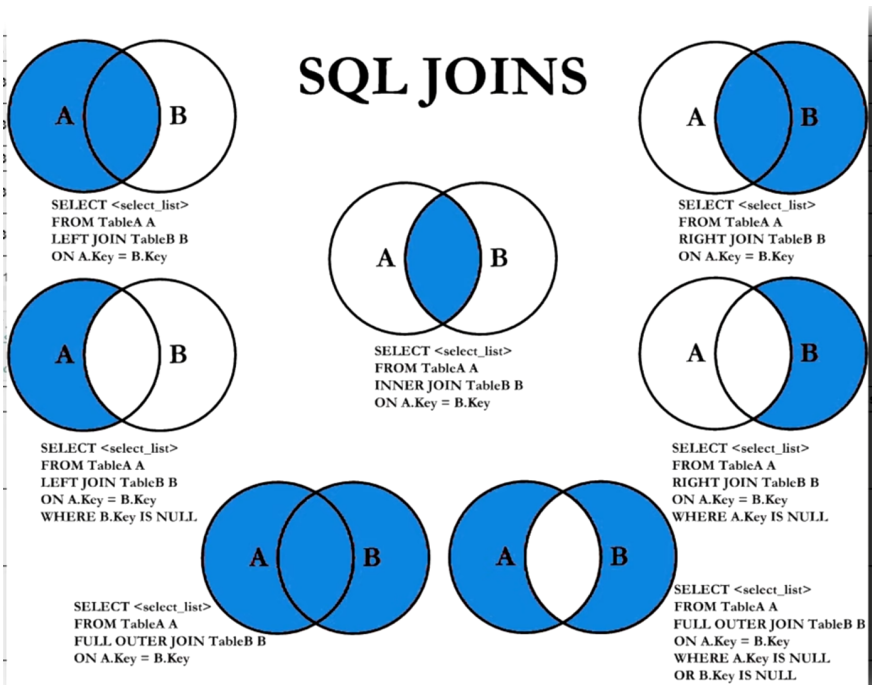
DISTINCT:

Ключевое слово DISTINCT используется совместно с инструкцией SELECT для возврата только уникальных записей (без дубликатов).

JOIN:

Используется для комбинации двух и более таблиц

- INNER JOIN — возвращает записи, имеющиеся в обеих таблицах
- LEFT JOIN — возвращает записи из левой таблицы, даже если такие записи отсутствуют в правой таблице
- RIGHT JOIN — возвращает записи из правой таблицы, даже если такие записи отсутствуют в левой таблице
- FULL JOIN — возвращает все записи объединяемых таблиц
- CROSS JOIN — возвращает все возможные комбинации строк обеих таблиц
- SELF JOIN — используется для объединения таблицы с самой собой



Агрегатные функции:

COUNT(*)	Возвращает количество строк источника записей
COUNT	Возвращает количество значений в указанном столбце
SUM	Возвращает сумму значений в указанном столбце
AVG	Возвращает среднее значение в указанном столбце
MIN	Возвращает минимальное значение в указанном столбце
MAX	Возвращает максимальное значение в указанном столбце

WHERE & HAVING

В *HAVING* и только в нём можно писать условия по агрегатным функциям (SUM, COUNT, MAX, MIN и т. д.)
WHERE выполняется до формирования групп *GROUP BY*
Следующим этапом формируются группы, которые указаны в *GROUP BY.* После того как сформированы группы, можно накладывать условия на результаты агрегатных функций. И тут как раз наступает очередь *HAVING*: выполняются условия, которые вы задали.
Главное отличие *HAVING* от *WHERE* в том, что в *HAVING* можно наложить условия на результаты группировки, потому что порядок исполнения запроса устроен таким образом, что на этапе, когда выполняется *WHERE*, ещё нет групп, а *HAVING* выполняется уже после формирования групп.

UNION & UNION ALL:

Предложение/оператор UNION используется для комбинации результатов двух и более инструкций SELECT. При этом, возвращаются только уникальные записи.
Предложение UNION ALL также используется для объединения результатов двух и более инструкций SELECT. При этом, возвращаются все записи, включая дубликаты.

Синонимы (aliases):

Синонимы (aliases) позволяют временно изменять названия таблиц и колонок. "Временно" означает, что новое название используется только в текущем запросе, в БД название остается прежним.

Индексы:

Индексы - это специальные поисковые таблицы (lookup tables), которые используются движком БД в целях более быстрого извлечения данных. Проще говоря, индекс — это указатель или ссылка на данные в таблице.
Индексы ускоряют работу инструкции `SELECT` и предложения `WHERE`, но замедляют работу инструкций `UPDATE` и `INSERT`. Индексы могут создаваться и удаляться, не оказывая никакого влияния на данные.

Транзакция:

https://habr.com/ru/post/537594/
Транзакция - это единица работы или операции, выполняемой над БД. Это последовательность операций, выполняемых в логическом порядке. Эти операции могут запускаться как пользователем, так и какой-либо программой, функционирующей в БД.
Транзакция - это применение одного или более изменения к БД. Например, при создании/обновлении/удалении записи мы выполняем транзакцию. Важно контролировать выполнение таких операций в целях обеспечения согласованности данных и обработки возможных ошибок.
Транзакция - упорядоченное множество операций (группа запросов), переводящих базу данных из одного согласованного состояния в другое.

- атомарность (atomicity) — все операции транзакции должны быть успешно завершены. В противном случае, транзакция прерывается, а все изменения отменяются (происходит откат к предыдущему состоянию)
- согласованность (consistency) — состояние должно измениться в полном соответствии с операциями транзакции
- изоляция или автономность (isolation) — транзакции не зависят друг от друга и не оказывают друг на друга никакого влияния
- долговечность (durability) — результат завершенной транзакции должен сохраняться при поломке системы

Блокировки транзакций:

Блокировка - метод управления параллельными процессами, при котором объект БД не может быть модифицирован без ведома транзакции
По области действия:
Строчная блокировка - действуют только на одну строку таблицы базы данных, не ограничивая манипуляции над другими строками таблицы.

Гранулярная блокировка - действует на всю таблицу или всю страницу и все строки. Блокировка, ограничивающая манипуляции со страницей данных в таблице (набор строк, объединённый признаком совместного хранения) иногда называется *страничной*
Предикатная блокировка - действует на область, ограниченную предикатом. Обычно это блокировка по диапазону ключей. При такой блокировке для ключа или индекса указывается значение или диапазон значений, на которые распространяется блокировка. Такая блокировка (а также блокировка всей таблицы), помимо прочего, защищает от чтения фантомов и обеспечивает уровень изоляции транзакции Serializable.
По строгости:
Совместная блокировка - накладывается транзакцией на объект в случае, если выполняемая ей операция безопасна, то есть не изменяет никаких данных и не имеет побочных эффектов. При этом, все транзакции могут выполнять операцию того же типа над объектом, если на него наложена совместная блокировка, обычно такая блокировка используется для операций чтения.
Исключительная блокировка - накладывается транзакцией на объект в случае, если выполняемая ей операция изменяет данные. Только одна транзакция может выполнять подобную операцию над объектом, если на него наложена исключительная блокировка. Блокировка не может быть наложена на объект, если на него уже наложена совместная блокировка.
По логике реализации:
Пессимистическая блокировка - накладывается перед предполагаемой модификацией данных на все строки, которые такая модификация предположительно затрагивает. Во время действия такой блокировки исключена модификация данных из сторонних сессий, данные из блокированных строк доступны согласно уровню изолированности транзакции. По завершению предполагаемой модификации гарантируется непротиворечивая запись результатов.
Оптимистическая блокировка - не ограничивает модификацию обрабатываемых данных сторонними сессиями, однако перед началом предполагаемой модификации запрашивает значение некоторого выделенного атрибута каждой из строк данных (обычно используется наименование `VERSION` и целочисленный тип с начальным значением 0). Перед записью модификаций в базу данных перепроверяется значение выделенного атрибута, и если оно изменилось, то транзакция откатывается или применяются различные схемы разрешения коллизий. Если значение выделенного атрибута не изменилось — производится фиксация модификаций с одновременным изменением значения выделенного атрибута (например, инкрементом) для сигнализации другим сессиям о том, что данные изменились.

Подзапросы:

```sql
SELECT * FROM users
WHERE userId IN (
  SELECT userId FROM users
  WHERE status = 'active'
);
```

Представления (view):

Представления - таблицы, чье содержание выбирается или получается из других таблиц. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных.
Обратите внимание: обновление строки в представлении приводит к ее обновлению в базовой таблице.

Обратите внимание: удаление строки в представлении приводит к ее удалению в базовой таблице.

Обобщенные табличные выражения / Common Table Expression (CTE):

http://www.sql-tutorial.ru/ru/book_common_table_expressions_cte/page1.html
Используются вместо *view* когда нет необходимости сохранять в метаданных базы его определение.
Табличные выражения - результаты запроса, которые можно использовать множество раз в других запросах. То есть, запросом мы достаем данные, и они помещаются в пространство памяти, аналогично временному представлению, которое физически не сохраняется в виде объектов. Далее мы работаем с получившейся конструкцией как с таблицей, используя такие конструкции как select, update, insert и delete.

```
1.  WITH Inc_Out AS (
2.    SELECT inc, 'inc' type, date, point
3.    FROM Income
4.    UNION ALL
5.    SELECT inc, 'inc' type, date, point
6.    FROM Income_o
7.    UNION ALL
8.    SELECT out, 'out' type, date, point
9.    FROM Outcome_o
10.   UNION ALL
11.   SELECT out, 'out' type,date, point FROM Outcome )
12. SELECT inc AS max_sum, type, date, point
13. FROM Inc_Out WHERE inc >= ALL ( SELECT inc FROM Inc_Out);
```

Django

```
# Секретный ключ
python -c 'from django.core.management.utils import get_random_secret_key;
print(get_random_secret_key())'
# Активация переменных окружения из .env
export $(cat .env | xargs)
# Запуск сервера на WSL
python manage.py runserver 0.0.0.0:8080
```

MVC Django:

MVC	—	*Django:*
Model	—	Model
View	—	Template
Controller	—	View

Запросы:

get_object_or_404 - получение одного значения или 404

get_list_or_404 - получение списка значений или 404
question.choice_set.all() - При создании Foreign Key в Choice Django автоматически генерирует в таблице Question поле для каждого экземпляра Choice. Мы можем обращаться к зависимым данным Choice через Question.
(https://stackoverflow.com/questions/2048777/what-is-choice-set-in-this-django-app-tutorial)

blank=True & null=True:

null=True - ячейка в БД может быть NULL
blank=True - форма или поле ввода данных может быть пустой

Удаление миграций

rm -r books/migrations - удалить миграции

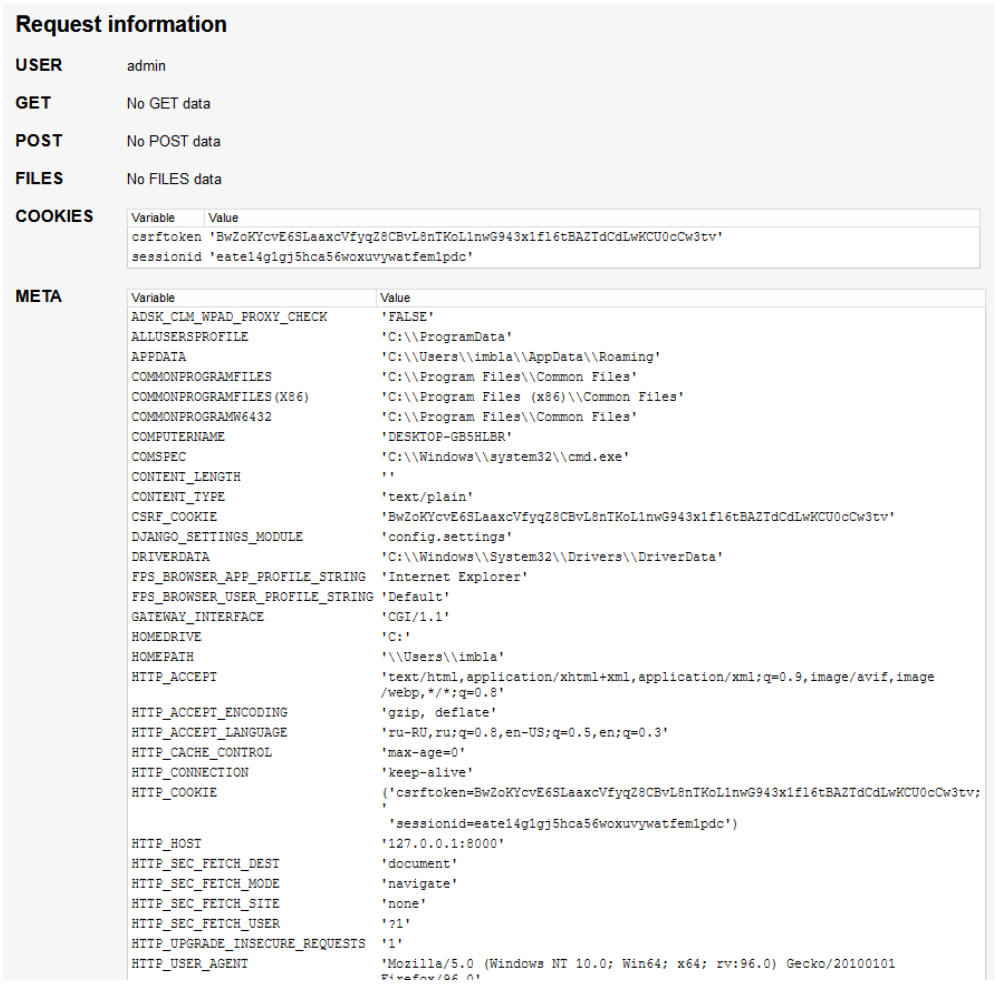
Django Rest Framework

HTTP COOKIES from request

```
class AnswerSerializer(serializers.ModelSerializer):
    ip = serializers.SerializerMethodField()

    def get_ip(self, obj):
        x_forwarded_for =
self.context['request'].META.get('HTTP_COOKIE')
        ip = x_forwarded_for.split(';')[0]
        return ip

    class Meta:
        model = Answer
        fields = ('id', 'question', 'text', 'user', 'date', 'ip')
```



Request information

USER admin

GET No GET data

POST No POST data

FILES No FILES data

COOKIES

Variable	Value
csrftoken	'BwZoKYcvE6SLaaxcVfyqZ8CBvL8nTKoL1nwG943x1fl6tBAZTdCdLwKCU0cCw3tv'
sessionid	'eate14g1gj5hca56woxuvywatfem1pdc'

META

Variable	Value
ADSK_CLM_WPAD_PROXY_CHECK	'FALSE'
ALLUSERSPROFILE	'C:\\ProgramData'
APPDATA	'C:\\Users\\imbla\\AppData\\Roaming'
COMMONPROGRAMFILES	'C:\\Program Files\\Common Files'
COMMONPROGRAMFILES(X86)	'C:\\Program Files (x86)\\Common Files'
COMMONPROGRAMW6432	'C:\\Program Files\\Common Files'
COMPUTERNAME	'DESKTOP-GB5HLBR'
COMSPEC	'C:\\Windows\\system32\\cmd.exe'
CONTENT_LENGTH	''
CONTENT_TYPE	'text/plain'
CSRF_COOKIE	'BwZoKYcvE6SLaaxcVfyqZ8CBvL8nTKoL1nwG943x1fl6tBAZTdCdLwKCU0cCw3tv'
DJANGO_SETTINGS_MODULE	'config.settings'
DRIVERDATA	'C:\\Windows\\System32\\Drivers\\DriverData'
FPS_BROWSER_APP_PROFILE_STRING	'Internet Explorer'
FPS_BROWSER_USER_PROFILE_STRING	'Default'
GATEWAY_INTERFACE	'CGI/1.1'
HOMEDRIVE	'C:'
HOMEPATH	'\\Users\\imbla'
HTTP_ACCEPT	'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'
HTTP_ACCEPT_ENCODING	'gzip, deflate'
HTTP_ACCEPT_LANGUAGE	'ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3'
HTTP_CACHE_CONTROL	'max-age=0'
HTTP_CONNECTION	'keep-alive'
HTTP_COOKIE	('csrftoken=BwZoKYcvE6SLaaxcVfyqZ8CBvL8nTKoL1nwG943x1fl6tBAZTdCdLwKCU0cCw3tv; ' 'sessionid=eate14g1gj5hca56woxuvywatfem1pdc')
HTTP_HOST	'127.0.0.1:8000'
HTTP_SEC_FETCH_DEST	'document'
HTTP_SEC_FETCH_MODE	'navigate'
HTTP_SEC_FETCH_SITE	'none'
HTTP_SEC_FETCH_USER	'?1'
HTTP_UPGRADE_INSECURE_REQUESTS	'1'
HTTP_USER_AGENT	'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:96.0) Gecko/20100101 Firefox/96.0'

SELECT & MULTISELECT:

https://www.django-rest-framework.org/api-guide/fields/#choice-selection-fields

Git

pass

DOCKER

Команды:

docker inspect ‘container_name’ — информация о контейнере
docker diff ‘container_name’ — список измененных файлов в рабочем контейнере
docker logs ‘container_name’ — все события произошедшие в контейнере
docker rm ‘container_name’ –- удалить контейнер
docker compose -f ‘docker-compose.yml’ –- запуск определенного файла

docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' 696eb9b29a02
–- ip адрес контейнера по его ID
docker exec -it ‘postgres-docker’ bash — зайти в контейнер с БД
psql -U postgres –- работать в БД

<\l> — все базы данных
<*dt*> — все таблицы

docker-compose exec web python <script.py> –- запуск скрипта в контейнере

docker ps
-a — список всех контейнеров
-q — список работающих контейнеров

docker run
-i — это сокращение для --interactive. Благодаря этому флагу поток STDIN. поддерживается в открытом состоянии даже если контейнер к STDIN не подключен.
-t — это сокращение для --tty. Благодаря этому флагу выделяется псевдотерминал, который соединяет используемый терминал с потоками STDIN и STDOUT контейнера. Для того чтобы получить возможность взаимодействия с контейнером через терминал нужно совместно использовать флаги -i и -t.
-d — запуск контейнера в фоновом режиме. (если появится ошибка мы не узнаем)
-rm — удалить контейнер после выхода.
-link — ссылка на другой контейнер.

Удаление тома БД:
docker volume ls –- сущ. тома
docker volume rm books_postgres_data –- удаление тома

CURL
Win
Для POST запроса с JSON:
>>>curl -X POST -H "Content-Type: application/json" -d "{""""links""""": [""""dom.com""""]}" “http://127.0.0.1:8000/api/visited_links”

React
Удалить React
npm uninstall -g create-react-app
npx clear-npx-cache

ClickHouse
Основные команды
показать доступные БД
SHOW DATABASES
показать таблицы БД
SHOW TABLES
переключиться на другую БД
USE <database_name>
Запуск CH в докере с портом 9000докер=9000win
docker run -d -p 9000:9000 clickhouse/clickhouse-server

Подключение к CH в докере из WSL
clickhouse-client -h 172.31.240.1
Права пользователя
access_management setting to 1 in users.xml (in container)

Testing

Django

Тестирование используемого шаблона

```
class HomePageTest(TestCase):


    def test_home_page_returns_correct_html(self):
        response = self.client.get('/')
        self.assertTemplateUsed(response, 'home.html')
```

Тестирование POST запроса

```
    def test_can_save_a_POST_request(self):
        response = self.client.post('/', data={'item_text': 'A new list
item'})
        self.assertIn('A new list item', response.content.decode())
        self.assertTemplateUsed(response, 'home.html')
```

Паттерны/шаблоны проектирования

Паттерн - часто встречающееся решение определенной проблемы при проектирование архитектуры программ. Высокоуровневое описание решения.

Зачем нужны паттерны:

- проверенные решения
- стандартизация кода
- общий программистский словарь

Основные группы паттернов:

- порождающие паттерны - гибкое создание объектов без внесения в программу лишних зависимостей
- структурные паттерны - способы построения связей между объектами
- поведенческие паттерны - эффективная коммуникация между объектами

Прототип (prototype)

Порождающий паттерн проектирования, позволяет копировать объекты, не вдаваясь в подробности их реализации
Использование *deepcopy*
Описание - https://refactoring.guru/ru/design-patterns/prototype
Код - https://github.com/pkolt/design_patterns/blob/master/generating/prototype.py
Код2 - https://refactoring.guru/ru/design-patterns/prototype/python/example#

Фасад (facade)
Структурный паттерн проектирования, который предоставляет простой (но урезанный) интерфейс к сложной системе объектов, библиотеке или фреймворку
Божественный объект - объект, который хранит в себе “слишком много” и делает “слишком много”.
Описание - https://refactoring.guru/ru/design-patterns/facade
Код - https://github.com/pkolt/design_patterns/blob/master/structural/facade.py
Код2 - https://refactoring.guru/ru/design-patterns/facade/python/example

Адаптер (adapter)
pass

Наблюдатель (observer)
Поведенческий паттерн проектирования, который создает механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах
Описание - https://refactoring.guru/ru/design-patterns/observer
Код - https://github.com/pkolt/design_patterns/blob/master/behavior/observer.py
Код2 - https://refactoring.guru/ru/design-patterns/observer/python/example

Machine Learning
Обучение с учителем - преобразует входные данные в набор данных, который прежде не был известен

Обучение без учителя - преобразует входные данные в набор сгруппированных данных (кластеризация)

Параметрическое обучение - построение прогноза (преобразование данных) происходит на основе модели, имеющей некоторые *параметры*, которые корректируются на основание сравнения прогноза и реального результата (реальных данных)
Число параметров предопределено человеком и не зав исит от данных

Непараметрическое обучение - построение прогноза (преобразование данных) происходит на основе модели, имеющей некоторые *параметры*, количество которых зависит от входных данных

DEPLOY
Паттерн - часто встречающееся решение определенной проблемы при проектирование

Staticfiles
Не требуется (!пока что вроде бы!) создавать отдельную папку для media (картинок)
Для статичный файлов в Django надо прописать в nginx.conf путь к файлам статики:

```
# подключаем статические файлы
    location /static/ {
        alias /code/staticfiles/;
    }
```

В файле docker-compose также надо прописать путь к файлам статики

```
    ...
    volumes:
      - .:/code
      - static_volume:/code/staticfiles
    ...
```