[**1.1. Вопросы на основе содержания общепрофессиональных и профессиональных дисциплин направления подготовки 09.03.03 «Прикладная информатика»**](#_9ucv5asv9t4h) **4**

[1. Системные и локальные шины. Организация ввода/вывода в вычислительной системе.](#_whu381wv7xj) 4

[2. Иерархия запоминающих устройств. Характеристики, принципы построения и функционирования запоминающих устройств. Организация виртуальной памяти.](#_mj1b8620d7a7) 6

[3. Основные типы и характеристики вычислительных систем.](#_pbflqj4vfhae) 7

[4. Организация автоматической работы ЭВМ. Управляющие функции процессора. Общая организация выполнения программы на ЭВМ.](#_3kmx59pansj) 8

[5. Принципы фон-Неймана по построению вычислительных систем.[КВ1]](#_1vir5u2yeez1) 9

[6. Общая характеристика языка программирования Python. Схема выполнения. Система типов. Сравнение с другими языками.](#_wh3qlw6lb4a) 10

[7. Списки в Python. Назначение, особенности, создание, изменение. Основные операции и методы. Способы итерации по спискам.](#_qsmkxeovqmfz) 11

[8. Словари в Python. Назначение, особенности, создание, изменение. Основные операции и методы. Способы итерации по словарям.](#_1vq7tat9frxk) 14

[Перебор элементов словаря:](#_3dy6vkm) 17

[9. Связные списки: однонаправленные и двунаправленные – принцип реализации. Сравнение скорости выполнения основных операций в связных списках и в динамическом массиве.](#_wegf3ja6zw7z) 17

[10. Основные алгоритмы сортировки. Их сложность, достоинства и недостатки. Алгоритм быстрого поиска в отсортированном массиве. Сложность поиска в отсортированном и не отсортированном массиве.[КВ2]](#_41nyxk1sbxbq) 19

[11. Основные технологии, использующиеся для разработки веб-приложений. Языки разметки и программирования, технологии, библиотеки и фреймворки.](#_oxbj9wayu1yd) 20

[12. Общая характеристика языка разметки HTML. Основные элементы - теги, атрибуты, значения. Структура веб-страницы. Единицы измерения и цвета в HTML.](#_etayw1stw35l) 21

[13. Общая характеристика языка CSS. Назначение, синтаксис, использование. Виды селекторов.](#_iz0lxlqcbxey) 22

[14. Общая характеристика языка JavaScript. Синтаксис, назначение, система типов. Среда исполнения. Особенности выполнения на клиенте и на сервере.](#_lblz7orljmj2) 23

[15. Серверное программирование. Особенности, обзор технологий и языков программирования. Распространенные серверные фреймворки.](#_s61x4qrj02rx) 24

[16. Технология взаимодействия клиентской и серверных частей веб-приложения. Протокол HTTP. Запросы и ответы.](#_mwwb5x8jhc5n) 25

[17. Адресация в интернете. Основные понятия о URI, URL, URN. Структура адресов, использование.3]](#_muekmqubv6o9) 26

[18. Эволюция методологий моделирования. Моделирование деятельности и моделирование процессов. Использование методологии ARIS для моделирования деятельности предприятия. Модели, атрибуты моделей, действия над моделями, типы моделей. Объекты, свойства объектов. Связи, свойства связей. Техническая реализация методологии моделирования.[КВ4]](#_env4pgh6rtvo) 28

[19. Разделение прав доступа и система безопасности современных многопользовательских операционных систем на примере Linux. Пользователи и группы, основные права. Основные команды для управления правами и пользователями.](#_3f4vz1ml6aza) 30

[20. Работа в командной строке bash. Основные понятия и структура команды. Основные команды для перемещения по каталогам, просмотра файлов и информации о содержимом файлов.](#_cio10zd4bzu2) 32

[21. Настройка и диагностика сетевых подключений в современных операционных системах на примере Linux. Удаленный доступ к операционной системе по протоколу SSH.](#_jccm6tstt6gk) 33

[22. Основные сетевые службы. Веб-технологии - необходимое программное обеспечение, назначение схема работы, основные протоколы.](#_2003b1agmco9) 34

[Основные сетевые службы](#_z337ya) 34

[Веб-технологии - необходимое программное обеспечение, назначение схема работы, основные протоколы.](#_3j2qqm3) 35

[23. Общая характеристика языка программирования C#. Схема выполнения, синтаксис, система типов.](#_c5135tqirinb) 38

[24. Объектно-ориентированное программирование в C#. Принципы, особенности реализации. Синтаксис.](#_o1x3yhsutbmv) 39

[25. Фреймворк .NET. Назначение, основные понятия, применение. Объектно-реляционное отображение (ORM).](#_uiug2ek1rldd) 42

[Как работает ОRМ?](#_yxp0cclnmobv) 43

[**1.3. Вопросы на основе содержания дисциплин профиля «ИТ-сервисы и технологии обработки данных в экономике и финансах»**](#_xkz4b8qr1kax) **43**

[1. Понятие, классификация и принципы построения банковских информационных систем (БИС).](#_3v02qvtwq1) 43

2[. Реализация решения задач кредитного модуля банковских информационных систем.](#_sltv4pyxkq04) 45

[3. Применение технологии Big Data в банковской сфере](#_6di90ls1gpc7) 46

[4. Системы дистанционного банковского обслуживания.](#_7ib3q702kq7) 48

[5. Информационные системы бухгалтерского учета (ИС БУ), их структура и классификация.](#_7wlyeoi440ec) 48

[6. Подходы к классификации бухгалтерских информационных систем, их и различия.](#_c4qz4wu312xy) 49

[7. Модель системы счетов как основа любой бухгалтерской информационной системы. Модель организации синтетического учета. Модели организации аналитического учета.[КВ2]](#_wgoa9m7tjwtf) 51

[8. Понятие базы данных. Виды баз данных. Проектирование баз данных.](#_u0g0d3wnmhu4) 52

[9. Общая характеристика языка запросов SQL. Декларативная парадигма. Основные операторы, виды операторов.](#_qe3109ceiwty) 53

[10. Оператор SELECT в SQL. Общий синтаксис, наиболее распространенные блоки. Условия, сортировки, ограничения. Примеры.](#_ou4de8wdlam6) 54

[11. Оператор JOIN в SQL. Общий синтаксис. Виды объединений, их различия и области применений. Примеры.](#_sxz89dz3685l) 55

[12. Использование групповых функций в SQL. Синтаксис оператора SELECT при использовании группировок. Примеры использования.](#_8ejjmxik700s) 57

[Список](#_a0s6nv3q6zs3) 58

[13. Реляционные базы данных. Нормализация отношений. Первая нормальная форма. Вторая нормальная форма. Третья нормальная форма. Другие нормальные формы.](#_mqg3gzp6teg) 58

[14. Транзакции в SQL. Понятие транзакции, организация транзакций, вложенные транзакции. Транзакции и управление параллельной работой.[КВ3]](#_n3clvtni8trc) 60

[15. Типы задач машинного обучения: классификация, регрессия, кластеризация. Обучение с учителем и без учителя. Машинное обучение как оптимизационная задача.](#_4ucfdcc7thsc) 61

[С учителем](#_2p2csry) 62

[Без учителя](#_147n2zr) 62

[16. Регрессия как задача машинного обучения. Постановка задачи, математическая формализация. Целевая функция. Примеры.](#_8sincf6jw6ht) 63

[17. Классификация как задача машинного обучения. Постановка задачи, математическая формализация. Целевая функция. Примеры.](#_rayuc0lphcwf) 64

[18. Метрики качества результатов машинного обучения. Метрики качества моделей регрессии и классификации. MAE, MSE, Precision, Recall, F1, ROC-AUC.](#_u9sa84y2x3jf) 65

[19. Кластеризация как задача машинного обучения. Метод ближайших соседей (kNN) и его обобщения. Подбор числа k. Методы метрической классификации в библиотеке Scikit-Learn.](#_p8iifxaniad7) 67

[Метод k-ближайших соседей](#_gtsbqcdtfnlc) 68

[20. Структура проекта нативного Android приложения. Основные файлы. Манифест приложения. Файлы с разметкой. Файлы ресурсов. Использование альтернативных ресурсов. Используемые языки для разработки и описания приложения.](#_sytqtafbyqzn) 69

[Доступ к ресурсам](#_vx1227) 72

[Основные языки программирования](#_3fwokq0) 72

[21. Компоненты Android приложений. Основные виды компонентов, их особенности и назначение. Жизненный цикл компонентов и использование методов жизненного цикла.](#_3m62n5wij890) 72

[Основные методы жизненного цикла приложения](#_ffodxhcoqihu) 74

[22. Хранение данных в Android приложении. Виды хранилищ данных, их особенности и использование из приложения.](#_l0a3kpha86dw) 74

[23. Виды мобильных приложений - нативные, веб, гибридные. Сравнительная характеристика. Используемые технологии и языки программирования.[КВ5]](#_v5eqq4zed62r) 77

[24. Библиотеки Python для обработки больших данных. Обработка данных, превышающих объем доступной памяти. Библиотека Dask.](#_9rekfu13zd3m) 78

[25. Библиотеки Python для обработки данных. Numpy, pandas - основные понятия, возможности, назначение, примеры использования. Основные форматы структурированных данных.](#_mai96duki14) 79

# 1.1. Вопросы на основе содержания общепрофессиональных и профессиональных дисциплин направления подготовки 09.03.03 «Прикладная информатика»

## 1. Системные и локальные шины. Организация ввода/вывода в вычислительной системе.

Для любой вычислительной системы, состоящей из множества подсистем, необходимо обеспечить быстрый и эффективный обмен данными между ними. Шина – это канал пересылки данных, используемый совместно различными блоками системы. Основными характеристиками шин являются разрядность передаваемых данных и скорость передачи данных.

**Системная шина (FSB)** – канал (магистральный), по которому процессор соединен с другими устройствами компьютера (памятью, видеокартой, жестким диском и так далее).

Непосредственно к системной шине подключен только CPU, остальные устройства подсоединяются к ней через специальные контроллеры, сосредоточенные в основном в северном мосте набора системной логики (чипсета) материнской платы.

Микропроцессор через системную шину подключается к системному контроллеру (обычно такой контроллер называют «северным мостом» North Bridge). Системный контроллер включает в себя контроллер оперативной памяти и контроллеры шин, к которым подключаются периферийные устройства. К северному мосту обычно подключают наиболее производительные периферийные устройства (например, видеокарты), а менее производительные устройства (микросхема BIOS, устройства с шиной PCI) подключаются к «южному мосту» (South Bridge), который соединяется с северным мостом специальной высокопроизводительной шиной. Набор из «южного» и «северного» мостов называют чипсетом (chipset). Системная шина работает в качестве магистрального канала между процессором и чипсетом.

Для согласования интерфейсов периферийные устройства подключаются к шине через контроллеры и порты.

Контроллер – специализированное устройство (или плата), управляющее работой некоторого периферийного устройства и обеспечивающее его связь с системной платой. Например, контроллер клавиатуры или жёсткого диска.

Адаптер – устройство, обеспечивающее согласование параметров входных и выходных сигналов в системе. Например, видеоадаптер, преобразующий цифровое изображение для отображения на аналоговом мониторе; адаптеры последовательного и параллельного портов.

Порты устройств – электронные схемы, содержащие один или несколько регистров ввода-вывода и позволяющие подключать периферийные устройства компьютера к внешним шинам процессора.

Последовательный порт обменивается данными с процессором побайтно, а с внешними устройства побитно, а параллельный порт получает и посылает данные побайтно.

К последовательному порту подключают медленно действующие или достаточно удаленные устройства (мышь).

К параллельному порту подсоединяют более быстрые устройства (принтер, сканер).

**Локальная шина (local bus)** — функционально специализированный вид шины в IBM PC для связи процессора с отдельными видами периферийных устройств (контроллеров накопителей, видеоадаптеров и т.п.).

Для увеличения производительности используют **локальные шины**, связывающие процессор непосредственно с контроллерами периферийных устройств. Наиболее известные – VL-bus, PCI и PCI-Express.

Высокоскоростные последовательные шины USB – служит для одновременного подключения большого количества внешних устройств – до 127.

Передача информации от периферийного устройства в ядро ЭВМ называется операцией ввода, а передача из ядра ЭВМ в периферийное устройство - операцией вывода. Связь устройств ЭВМ друг с другом осуществляется с помощью средств сопряжения - интерфейсов.

Интерфейс представляет собой совокупность линий и шин, сигналов, электронных схем и алгоритмов, предназначенную для осуществления обмена информацией между устройствами. От характеристик интерфейсов во многом зависят производительность и надежность вычислительной машины.

Для обеспечения параллельной работы процессора и периферийных устройств схемы управления вводом-выводом отделяют от процессора. Выполнение общих функций возлагают на общие для групп периферийного оборудования унифицированные устройства – контроллеры прямого доступа к памяти, процессоры ввода-вывода.

К операциям ввода-вывода относятся все операции, управляющие работой внешних устройств и обеспечивающие передачу данных между внешними устройствами и оперативной памятью. Операции ввода-вывода распространяются и на внешние запоминающие устройства, и на устройства ввода-вывода, которые, в связи с этим принято, относить к одному классу устройств – внешним (периферийным) устройствам.

## 2. Иерархия запоминающих устройств. Характеристики, принципы построения и функционирования запоминающих устройств. Организация виртуальной памяти.

Иерархия запоминающих устройств, чем ниже - тем больше количество памяти и большее время доступа

1. Регистры процессора, память куда ложаться данные для текущей операции процессора, размер до нескольких тысяч байт, время - 1 такт
2. Кэш процессора (имеет подуровни, количество варьируется от процессора), данные из ОЗУ для быстрого доступа процессором
   1. L1 - десятки килобайт, до 10 тактов
   2. L2 - десятки мегабайт, до 100 тактов
   3. L3 - до сотни мегабайт. сотни тактов
   4. L4 - сотни мегабайт
3. Оперативная память (RAM, ОЗУ), до 1000 тактов
4. Внешняя память
   1. SSD (твердотельные накопители, flash)
   2. HDD (жесткий диск)
   3. Магнитная (магнитные ленты, дискеты)

1 - 3 пункты - энергозависимая память, то есть при отключении питания данные удаляются

**Виртуальная память** — это подход к управлению памятью компьютером, который скрывает физическую память (в различных формах: оперативная память, ПЗУ или жесткие диски) за единым интерфейсом, позволяя создавать программы, которые работают с ними как с единым непрерывным массивом памяти с произвольным доступом.

**1) Страничная организация виртуальной памяти**

Организуется с помощью страничной адресации. Оперативная память делится на страницы: области памяти фиксированной длины, которые являются минимальной единицей выделяемой памяти - виртуальные страницы.

Часть виртуальных страниц размещается в ОЗУ, а часть во внешней памяти. Место на жестком диске, где размещаются виртуальные страницы называют файлом подкачки или страничным файлом.

**2) Сегментная организация виртуальной памяти**

Виртуальное пространство делится на части произвольного размера — сегменты. Позволяет разбить данные процесса на логические блоки. Для каждого сегмента, как и для страницы, могут быть назначены права доступа к нему пользователя и его процессов. При загрузке процесса часть сегментов помещается в оперативную память (для каждого из сегментов ОС подыскивает подходящий участок свободной памяти), а часть сегментов размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки. Во время загрузки система создает таблицу сегментов процесса, в которой для каждого сегмента указывается начальный физический адрес сегмента в оперативной памяти, размер сегмента, правила доступа, признак модификации, признак обращения к данному сегменту за последний интервал времени и некоторая другая информация.

## 3. Основные типы и характеристики вычислительных систем.

Под **вычислительной системой** **(ВС)** понимают совокупность взаимосвязанных и взаимодействующих процессоров или ЭВМ, периферийного оборудования и программного обеспечения, предназначенную для сбора, хранения, обработки и распределения информации.

Отличительной особенностью ВС по отношению к ЭВМ является наличие в них нескольких вычислителей, реализующих параллельную обработку. Создание ВС преследует следующие основные цели: повышение производительности системы за счет ускорения процессов обработки данных, повышение надежности и достоверности вычислений, предоставление пользователям дополнительных сервисных услуг и т.д

По назначению:

* **универсальные** - для широкого спектра задач
* **специализированные** - решение узкого класса задач

По типу построения:

* **многомашинные** - на базе нескольких компьютеров
* **многопроцессорные** - на базе нескольких процессоров

По типу ЭВМ или процессоров:

* **Однородная ВС** строится на базе однотипных компьютеров или процессоров, позволяют использовать стандартные наборы технических, программных средств, стандартные протоколы (процедуры) сопряжения уст­ройств. Поэтому их организация значительно проще, облегчается обслуживание систем и их модернизация.
* **Неоднородная ВС** включает в свой состав различные типы компьютеров или про­цессоров. При построении системы приходится учитывать их различные техни­ческие и функциональные характеристики, что существенно усложняет создание и обслуживание неоднородных систем.

По методам управления элементами ВС различают:

* **централизованные**
* **децентрализованные**
* **со смешанным управлением.**

По принципу закрепления вычислительных функций

* **Системы с жестким закреплением функций** - для статического размещения программных модулей и массивов данных
* **Системы с плавающим закреплением функций** - для динамического размещения программных модулей и массивов данных

По степени территориальной разобщенности вычислительных модулей ВС

* **территориально-сосредоточенные –** все компоненты располагаются в непо­средственной близости друг от друга;
* **распределенные –** компоненты могут располагаться на значительном расстоя­нии, например, вычислительные сети;
* **структурно-одноуровневые** **–** имеется лишь один общий уровень обработки данных;
* **многоуровневые (иерархические) структуры** **–** в иерархических ВС маши­ны или процессоры распределены по разным уровням обработки информации, некоторые машины (процессоры) могут специализироваться на выполнении определенных функций.

По режиму работы ВС работающие в:

* **в оперативном временном режиме,** используют режим реального масштаба времени. Этот режим характеризуется жесткими ограничениями на время решения задач в системе и предполагает высокую степень автоматизации процедур ввода-вывода и обработки данных.
* **неоперативном временном режиме**

## 4. Организация автоматической работы ЭВМ. Управляющие функции процессора. Общая организация выполнения программы на ЭВМ.

Процессор – устройство, предназначенное для автоматического выполнения аппаратно – микропрограммным способом набора операций, составляющих его систему команд.

Все операции, входящие в в систему команд условно делятся на следующие группы:

1. Команды пересылки

* внутри микропроцессора (move)
* команды ввода/вывода (in, out)

1. Арифметические операции (\*, /, -, +)
2. Логические операции
3. Операции передачи управления
   * безусловная передача управления
   * условная передача управления
4. Команды передачи управления
5. Команды управления (задача супервизора, выполнить асинхронные вычисления и т.д.)
6. Специальные команды (работа со списками, с цепочками данных, умножение со сложением, умножение с вычитанием)

**Цикл работы процессора** – действия процессора по исполнению одной команды. Он включает в себя:

1. Выборку фрагмента программы из Оперативной Памяти (ОП), распаковку команды и размещение ее в регистре команд.
2. Формирование исполнительного адреса операндов и проверка адреса на корректность.
3. Выбор операндов из ОП или регистровой памяти.
4. Выполнение операции указанной в операционной части команды (КОП).
5. Обработка прерывания.
6. Запись результата.

Действия по выполнению цикла работы процессора выполняются каждым процессором, но отличаются друг от друга, в зависимости от структуры и архитектуры компьютера. В отдельных процессорах составляющие цикла работы совмещены во времени.

## 5. Принципы фон-Неймана по построению вычислительных систем.[КВ1]

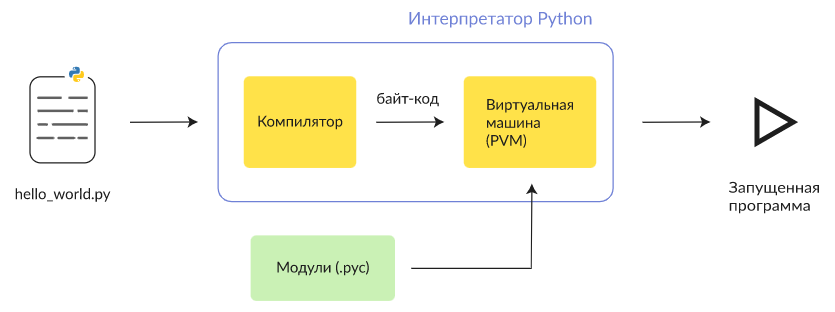
Современные ЭВМ построены в соответствии с принципами, сформулированными фон Нейманом в 1945 г.:

1. **Принцип программного управления**: ЭВМ работает по программе, которая находится в оперативной памяти и выполняется автоматически; программы дискретны и представляют собой последовательность команд, каждая из которых осуществляет отдельный акт преобразования информации; все разновидности команд образуют систему команд машины.
2. **Принцип условного перехода**: При выполнении программы возможен переход к той или иной команде в зависимости от промежуточных результатов вычислений; это допускает создание циклов.
3. **Принцип хранимой информации**: Команды, как и операнды, представляются в машинном коде и хранятся в оперативной памяти. При работе команды обрабатываются устройством управления процессора, а операнды - арифметико-логическим устройством.
4. **Принцип использования двоичной системы счисления**: Информация кодируется в двоичной форме и разделяется на элементы, называемыми словами. В двоичной системе используются две цифры 0 и 1, что соответствует двум состояниям двустабильной системы (кнопка нажата-отпущена, транзистор открыт-закрыт)
5. **Принцип иерархичности ЗУ**: Компромиссом между необходимой большой емкостью памяти, быстрым доступом к данным, дешевизной и надежностью является иерархия запоминающих устройств: 1) быстродействующее ОЗУ, имеющее небольшую емкость для операндов и команд, участвующих в вычислениях; 2) инерционное ВЗУ, имеющее большую емкость для информации, не участвующей в данный момент в работе ЭВМ.

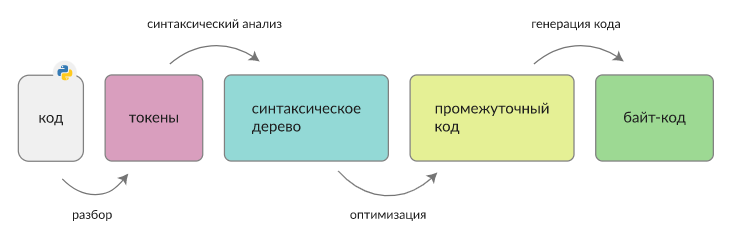
## 6. Общая характеристика языка программирования Python. Схема выполнения. Система типов. Сравнение с другими языками.

Python — интерпретируемый язык программирования. Он не конвертирует свой код в машинный, который понимает железо (в отличие от С и С++). Вместо этого, Python-интерпретатор переводит код программы в байт-код, который запускается на виртуальной машине Python (PVM).

Интерпретатор — это программа, которая конвертирует инструкции, написанные на Python, в байт-код и выполняет их. По сути интерпретатор — это программный слой между вашим исходным кодом и железом.



Компиляция:



К основным встроенным типам относятся:

1. *None* (неопределенное значение переменной)
2. Логические переменные (*Boolean Type*)
3. Числа (*Numeric Type*)
   1. *int* – целое число
   2. *float* – число с плавающей точкой
   3. *complex* – комплексное число
4. Списки (*Sequence Type*)
   1. *list* – список
   2. *tuple* – кортеж
   3. *range* – диапазон
5. Строки (*Text Sequence Type* )
   1. *str*
6. Бинарные списки (*Binary Sequence Types*)
   1. *bytes* – байты
   2. *bytearray* – массивы байт
   3. *memoryview* – специальные объекты для доступа к внутренним данным объекта через protocol buffer
7. Множества (*Set Types*)
   1. *set* – множество
   2. *frozenset* – неизменяемое множество
8. Словари (*Mapping Types*)
   1. *dict* – словарь

## 7. Списки в Python. Назначение, особенности, создание, изменение. Основные операции и методы. Способы итерации по спискам.

Списки в Python – упорядоченные изменяемые коллекции объектов произвольных типов.

**Создать** список можно несколькими способами:

1. Можно обработать итерируемый объект, например строку, встроенной функцией list

2. С помощью литерала (представление значения некоторого типа данных). Список может содержать любое количество любых объектов (в том числе и вложенные списки), или не содержать ничего.

3. Генераторы списков. Генератор списков – способ построить новый список, применяя выражение к каждому элементу последовательности. Генератор списков очень похож на цикл for

**Изменение**

Списки – это изменяемые объекты, поэтому их элементы могут изменяться, или же может меняться их порядок

Изменение элемента по индексу:

my\_list = ['один', 'два', 'три', 'четыре', 'пять']

my\_list[2] = 'ноль'

['один', 'два', 'ноль', 'четыре', 'пять']

Если индекс – отрицательное число, то он будет считаться с последнего элемента

Основные операции и методы списков.

Списки могут изменяться с помощью встроенных функций и методов списков.

Операции со списками

1. Получение среза (выборка подмножества элементов списка)

my\_list = ['один', 'два', 'три', 'четыре', 'пять']

print(my\_list[1:3])

['два', 'три']

2. Сложение списков.

print([1.3,3,4]+[5,4])

[1.3, 3, 4, 5, 4]

3. Применение агрегатных функций

print(len([1.3,3,4])) – длина списка

print(min(my\_list))

print(max(my\_list))

print(sum(my\_list))

4.Сравнение списков

a = [3,4]

b = [3,4]

print(a == b)

True

5.Повторение списка с помощью оператора умножения

a = [7,4]\*2

print(a)

[7, 4, 7, 4]

6.Проверка принадлежности элемента списка 3 in a, 3 not in a

7. Удаление элемента из списка: del a[0] (по индексу)

8.Получение значение элемента по индексу a[0]

*Методы списков*

1. list.append (x) – добавляет элемент в конец списка

2. list.extend(L) – Расширяет список list, добавляя в конец все элементы списка L

3. list.insert(i, x) – вставляет на i-ый элемент значение x.

4. list.remove(x) – удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует

5. list.pop([i]) – удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляет последний элемент

6. list.index(x, [[start[,end]]) – Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)

7. list.count(x) – возвращает количество элементов со значением x

8. list.sort([key=функция]) – сортирует список на основе функции

9. list.reverse() – разворачивает список

10. list.copy() – поверхностная копия списка

11. list.clear() – очищает список

Перебор элементов списка

1. Цикл for

2. Цикл while

3. Перебор с помощью генератора

arr = [1, 2, 3]

arr = [2\*i for i in arr]

## **8. Словари в Python. Назначение, особенности, создание, изменение. Основные операции и методы. Способы итерации по словарям.**

**Словари в Python** (dict) – это неупорядоченные коллекции произвольных объектов с доступом по ключу. Их также называют ассоциативными массивами или хеш-таблицами. Это структура данных, позволяющая идентифицировать ее элементы не по числовому индексу, а по произвольному. **Назначение**: при помощи словарей мы можем разработать эффективные структуры данных, в словарях можно содержать объединенные данные в виде записей. Также словари эффективны при представлении разреженных структур данных и для словарей нет необходимости писать алгоритмы поиска, т.к. эти операции уже реализованы.

Основные **особенности словарей** следующие:

* **в словарях доступ к элементам выполняется по ключу, а не по индексу.** Если в словаре реализовывать доступ к значению по индексу, то в этом случае индекс представляет собой ключ, а не смещение относительно начала;
* **словари представляют неупорядоченные коллекции произвольных объектов.** Порядок формирования элементов в словаре определяет интерпретатор.
* **словари имеют переменную длину.**
* **гетерогенность** – словари могут содержать объекты любых типов;
* **произвольное количество вложений** – могут содержать списки, другие словари и т.п.;
* **словари изменяемые объекты.**
* **словари являются таблицами ссылок на объекты** или хеш-таблицами и относятся к отображаемым объектам. Это означает, что в словарях объекты отображают (представляют) ключи на значение.

**Создание словарей**

1) С помощью литерала:

>>> d = {}

>>> d

{}

>>> d = {'dict': 1, 'dictionary': 2}

>>> d

{'dict': 1, 'dictionary': 2}

2) С помощью функции dict:

>>> d = dict(short='dict', long='dictionary')

>>> d

{'short': 'dict', 'long': 'dictionary'}

>>> d = dict([(1, 1), (2, 4)])

>>> d

{1: 1, 2: 4}

3) С помощью функции fromkeys

>>> d = dict.fromkeys(['a', 'b'])

>>> d

{'a': None, 'b': None}

>>> d = dict.fromkeys(['a', 'b'], 100)

>>> d

{'a': 100, 'b': 100}

4) С помощью генераторов словарей

>>> d = {a: a \*\* 2 for a in range(7)}

>>> d

{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}

**Изменение словарей**

1) Чтобы добавить элемент в словарь нужно указать новый ключ и значение.

>>> d1 = {"Russia":"Moscow", "USA":"Washington"}

>>> d1["China"]="Beijing"

>>> print(d1)

{'Russia': 'Moscow', 'China': 'Beijing', 'USA': 'Washington'}

2) Для удаления элемента из словаря можно воспользоваться командой del.

>>> d2 = {"A1":"123", "A2":"456"}

>>> del d2["A1"]

>>> print(d2)

{'A2': '456'}

**Основные операции**

1) Чтобы получить доступ к элементам словаря, нужно передать ключ в квадратных скобках []:

>>> dict\_sample = {"Company": "Toyota", "model": "Premio", "year": 2012}

>>> x = dict\_sample["model"]

>>> print(x)

Premio

2) Чтобы определить, существует ли заданный ключ в словаре, в Python используется операция in. Общая форма использования операции in следующая

f\_is = key in D (где D – исходный словарь, key – ключ, наличие которого в словаре D нужно определить, f\_is – результат логического типа. Если f\_is=True, то ключ key присутствует в словаре. Если f\_is=False, то ключа нету в словаре)

3) Операция not in возвращает результат всегда противоположный операции in. Общая форма операции not in следующая:

f\_is = key not in D

**Методы словарей**

**dict.len()** – количество элементов в словаре.

**dict.clear**() - очищает словарь.

**dict.copy**() - возвращает копию словаря.

classmethod

**dict.fromkeys**(seq[, value]) - создает словарь с ключами из seq и значением value (по умолчанию None).

**dict.get**(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).

**dict.items**() - возвращает пары (ключ, значение).

**dict.keys**() - возвращает ключи в словаре.

**dict.pop**(key[, default]) - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).

**dict.popitem**() - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.

**dict.setdefault**(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).

**dict.update**([other]) - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).

**dict.values**() - возвращает значения в словаре.

## **Перебор элементов словаря:**

1) Итерация по .items():

Объект представления, возвращаемый функцией .items(), выдает пары ключ-значение по одной и позволяет перебирать словарь, и таким образом, вы получаете доступ к ключам и значениям одновременно.

>>> for item in a\_dict.items():

... print(item)

...

('color', 'blue')

('fruit', 'apple')

('pet', 'dog')

2) Итерация через .keys()

Чтобы перебрать словарь в Python с помощью .keys(),нужно вызвать .keys() в заголовке цикла for:

>>> for key in a\_dict.keys():

... print(key)

...

color

fruit

pet

3) Итерация по .values()

Как и любой объект представления, объект, возвращаемый функцией .values(), также может быть итерирован. В этом случае .values() возвращает значения a\_dict:

>>> for value in a\_dict.values():

... print(value)

...

blue

apple

dog

## 9. Связные списки: однонаправленные и двунаправленные – принцип реализации. Сравнение скорости выполнения основных операций в связных списках и в динамическом массиве.

Связанные списки:

* Имеем только первый элемент списка
* Каждый элемент имеет указатель на следующий элемент списка
  + Для двунаправленного также хранится указатель на предыдущий элемент списка
* Обеспечивают последовательный доступ к данным

Принцип реализации

class Node:

next: Node = None

value: Any

previous: Node = None # для двусвязного

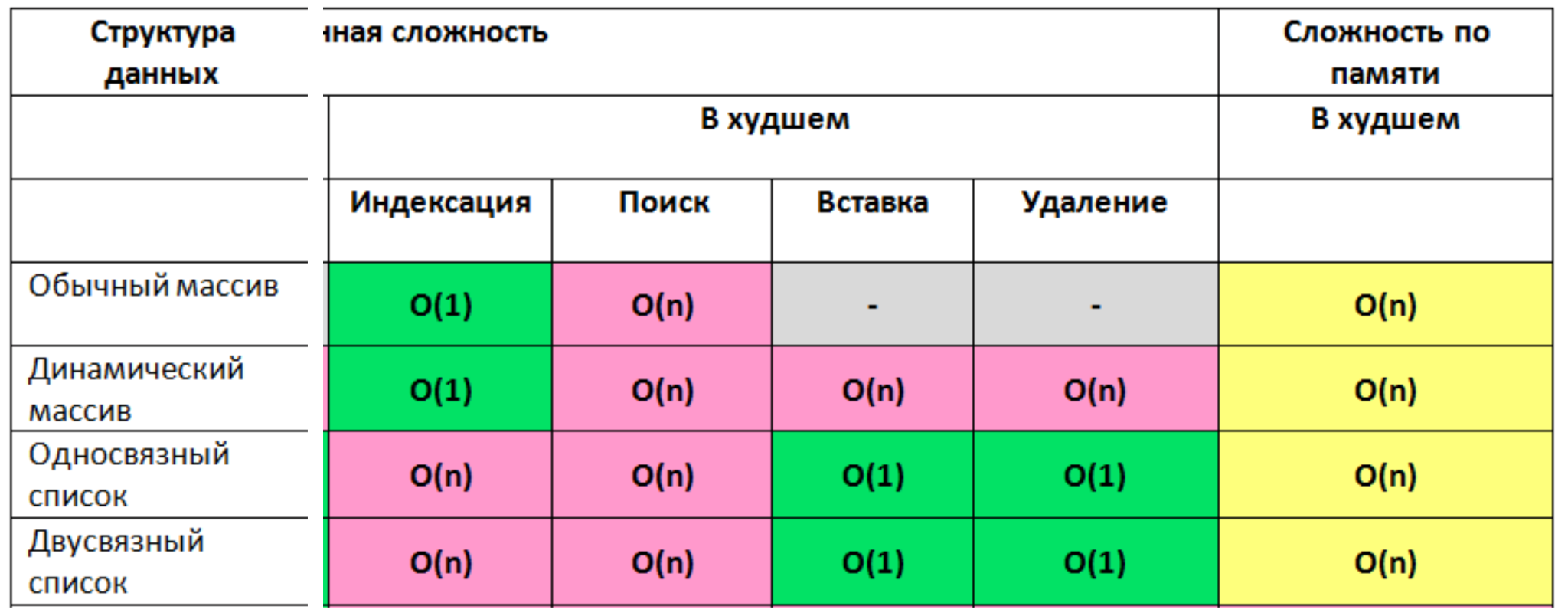
Преимущества

* эффективное (за константное время) добавление и удаление элементов
* размер ограничен только объёмом памяти [компьютера](https://ru.wikipedia.org/wiki/%D0%AD%D0%92%D0%9C) и разрядностью указателей
* динамическое добавление и удаление элементов

Недостатки

* сложность прямого доступа к элементу, а именно определения физического адреса по его [индексу](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_(%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2)) (порядковому номеру) в списке
* на поля-указатели (указатели на следующий и предыдущий элемент) расходуется дополнительная память (в [массивах](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2), например, указатели не нужны)
* некоторые операции со списками медленнее, чем с массивами, так как к произвольному элементу списка можно обратиться, только пройдя все предшествующие ему элементы
* соседние элементы списка могут быть распределены в памяти нелокально, что снизит эффективность [кэширования](https://ru.wikipedia.org/wiki/%D0%9A%D1%8D%D1%88) данных в процессоре
* над связными списками, по сравнению с массивами, гораздо труднее (хоть и возможно) производить параллельные векторные операции, такие, как вычисление суммы: накладные расходы на перебор элементов снижают эффективность распараллеливания
* при потере начального головного узла,теряется и весь список

Сложность



## 10. Основные алгоритмы сортировки. Их сложность, достоинства и недостатки. Алгоритм быстрого поиска в отсортированном массиве. Сложность поиска в отсортированном и не отсортированном массиве.[КВ2]

<https://academy.yandex.ru/posts/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii>

Алгоритмы сортировки представляют из себя последовательность действий, в результате которых последовательность элементов становится упорядоченной. Каждый такой алгоритм имеет свои плюсы и минусы, которые заключаются в количестве действий при разных изначальных конфигурациях последовательности. Можно сделать наиболее оптимальный выбор алгоритма, заранее зная данные, заключенные в массиве.

Основные алгоритмы сортировки: [Основные виды сортировок и примеры их реализации — Академия Яндекса](https://academy.yandex.ru/posts/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii) - с гифками и кодом

* Сортировка пузырьком – сравниваются пары элементов, при обнаружении неупорядоченности в рамках пары, элементы меняются местами, перебор идёт в одном направлении. О(n^2)
* Сортировка перемешиванием – пузырёк, но двунаправленный. O(n^2)
* Сортировка расчёсыванием – пызурёк, но с расстоянием между сравниваемыми элементами, которое постепенно уменьшается. Оптимальное расстояние – n/1,247, после каждого прохода расстояние делится на 1,247. Сложность, также, О(n^2), но, в среднем, может быть меньше.
* Сортировка вставками – каждый следующий элемент вставляется в нужную по порядку позицию в итоговый массив. O(n^2)
* Сортировка выбором – как вставки, только наоборот: в неотсортированной части ищется наибольший (или наименьший) элемент, после чего он извлекается, добавляясь в конец (начало) отсортированной части. Сложность O(n^2)
* Быстрая сортировка. В среднем, O(n log n). 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.
* Сортировка слиянием. Суть в том, что массив делится на два подмассива, каждый из них сортируется, после чего подмассивы сливаются в отсортированный. Данный алгоритм применяется рекурсивно к подмассивам. Сложность, даже в худшем случае, O(n log n).
* Пирамидальная сортировка – из элементов формируется двоичная куча, по которой и строится отсортированный массив. Сложность, даже в худшем случае, O(n log n).

Двоичный (бинарный) поиск (метод деления пополам, дихотомия) — классический алгоритм поиска элемента в **отсортированном массиве**, 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.

## 11. Основные технологии, использующиеся для разработки веб-приложений. Языки разметки и программирования, технологии, библиотеки и фреймворки.

Для написания frontend- части приложения преимущественно используется язык программирования JavaScript в связке с HTML (язык гипертекстовой разметки, говорит, какие элементы должны находиться на странице) и CSS (таблица стилей, управляем внешним видом и расположением элементов на странице).

Для удобства разработки используются JavaScript- фреймворки- такие, как Vue, Svelte, React, Angular. Каждый из них обладает своим конкурентным преимуществом для определенного типа задач.

Для SPA- приложений (веб-приложение или веб-сайт, использующий единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript) у каждого из фреймворков, перечисленных ранее, есть свои библиотеки для организации единого места хранения информации и навигации по странице- так называемые “сторы” и “роутеры”. Для примера, у Vue это библиотеки Vuex и Vue-router.

Мало какое Веб-приложение может работать без бекенда (за исключением одностраничных рекламных лендингов). Для написания backend- части веб-приложения существует много серверных языков программирования, но наиболее популярные из них- Python, Go, Java, PHP, Ruby. У каждого из них также есть свои библиотеки и фреймворки для более быстрой и удобной организации серверной архитектуры.

Для связи между фронтенд и бэкэнд частями приложения используется HTTP - протокол запросов клиентской части приложения к серверной. На данный момент самые распространенные способы организации API (код, который позволяет двум приложениям обмениваться данными с сервера)- Rest api и GraphQL.

## 12. Общая характеристика языка разметки HTML. Основные элементы - теги, атрибуты, значения. Структура веб-страницы. Единицы измерения и цвета в HTML.

**HTML (Hyper Text Markup Language)** - язык разметки, стандартизированный язык разметки документов в Интернете. Язык гипертекстовой разметки HTML был разработан британским учёным Тимом Бернерсом-Ли приблизительно в 1986—1991 годах в стенах ЦЕРНа в Женеве в Швейцарии. Актуальная версия HTML5 выпущена консорциумом W3C в 2014 году.

[Элементы HTML](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D1%8B_HTML) являются строительными блоками HTML страниц. С помощью HTML разные конструкции, изображения и другие объекты, такие как [интерактивная веб-форма](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0_(HTML)), могут быть встроены в отображаемую страницу. HTML предоставляет средства для создания заголовков, абзацев, списков, ссылок, цитат и других элементов. Элементы HTML выделяются тегами, записанными с использованием угловых скобок.

**Тег** (иногда тэг, англ. tag) в HTML — элемент языка разметки гипертекста, более правильное название — дескриптор. Например, текст википедии, заключенный между начальным тегом *<small>* и конечным тегом *</small>* предписывает отображать его меньше основного текста. Теги используются в паре — открывающий, или начальный, и закрывающий, или конечный, или еще дополнительно возможно применение одиночного тега и тега пустого элемента.

У тега могут быть свойства, называемые атрибутами, дающие дополнительные возможности форматирования текста. Они записываются в виде пары имя-значение, причем нечисловое значение заключается в кавычки. Например, можно выделить фрагмент текста определенным шрифтом (тег *<font>*), указав в этом теге название шрифта и желаемый размер: *<font face=“Times, Arial, Courier” size=4>оформляемый текст </font>*.

**Веб-страница** - это документ, информационный ресурс находящийся во всемирной паутине и доступ к которому осуществляется через веб-браузер.

Чаще всего веб-страница состоит из языка разметки HTML, который может содержать ссылки на файлы в других форматах, а также гиперссылки для быстрого перехода на другие веб-страницы или доступа к ссылочным файлам. Веб страницы могут быть статическими (файл лежащий на сервере и никак не меняющий свое наполнение) и динамическими (выводящие данные из БД и меняющие свое наполнения от итераций).

HTML применяется для разметки сырой страницы, для создания прототипа, потом подключается CSS наполняющий страницу графикой и оформлением, последний этап это JavaScript(или другой язык программирования способный взаимодействовать с документом HTML.), который можно использовать для изменения содержимого веб-сайта и заставить его вести себя по-разному в ответ на действия пользователя. Общее использование JavaScript включает в себя окна подтверждения, призывы к действию и добавление новых идентификаторов к существующей информации.

В коде HTML-документа есть возможность указывать размеры тех или иных объектов оформления Web-страницы, а также цветовые свойства этих объектов. Предусмотрено два способа задания цвета. Чаще всего используется способ с указанием RGB-кода требуемого цвета. Для часто встречаемых цветов есть буквенные обозначения.

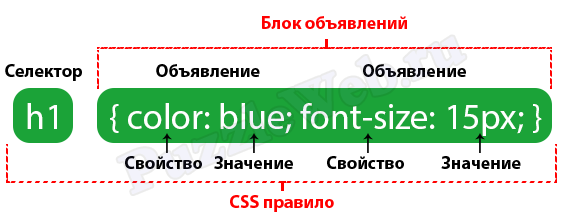
Указывать размеры каких-либо объектов оформления Web-страниц только двумя способами - абсолютные и относительные. Либо указывать их размер в пикселях, либо — в процентном отношении.

## 13. Общая характеристика языка CSS. Назначение, синтаксис, использование. Виды селекторов.

CSS (Cascading Style Sheets) — язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, документам HTML и приложениям XML).

Обычно CSS-стили используются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML, но также могут быть применены к любому виду XML-документ.

Синтаксис:



* Первым всегда указывается селектор, он сообщает браузеру, к какому элементу или элементам веб-страницы будет применен стиль.
* Далее следует блок объявлений, который начинается с открывающей фигурной скобки { и заканчивается закрывающей }, между фигурными скобками указываются форматирующие команды (объявления), которые используются браузером для стилизации выбранного селектором элемента.
* Каждое объявление состоит из двух частей: свойства и его значения. Объявление всегда должно заканчиваться точкой с запятой (;). Опустить указание ";" можно только в конце последнего объявления перед закрывающей фигурной скобкой.
* Свойство - это команда форматирования, определяющая конкретный стилевой эффект для элемента. Само свойство представляет из себя слово или несколько написанных через дефис слов. Каждое свойство имеет свой предопределенный набор значений. После имени свойства указывается двоеточие, которое отделяет название свойства от допустимого значения.

## 14. Общая характеристика языка JavaScript. Синтаксис, назначение, система типов. Среда исполнения. Особенности выполнения на клиенте и на сервере.

JavaScript — мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили.

Синтаксис языка JavaScript во многом напоминает синтаксис Си и Java

В JavaScript:

* все идентификаторы регистрозависимы,
* в названиях переменных можно использовать буквы, подчёркивание, символ доллара, арабские цифры,
* названия переменных не могут начинаться с цифры,
* для оформления однострочных комментариев используются //, многострочные и внутристрочные комментарии начинаются с /\* и заканчиваются \*/.

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

**Основные архитектурные черты**: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

Наиболее распространённой **средой исполнения** JavaScript является браузер. Как правило, браузеры предоставляют открытый API для создания объектов среды исполнения (англ. host objects), предоставляющих возможность работы с DOM в JavaScript.

Другой распространённой **средой исполнения** JavaScript является веб-сервер. Веб-сервер, поддерживающий JavaScript, предоставляет объекты среды исполнения, представляющие HTTP-запрос и HTTP-ответ. Манипулируя этими объектами, программа на JavaScript может динамически генерировать страницы. Например, технология ASP для веб-сервера IIS позволяет реализовывать серверную часть в том числе и на языке программирования JScript (реализация JavaScript от компании Microsoft). Другой пример - это веб-сервер Jaxer (англ.), предоставляющий помимо объектов, традиционных для серверов, еще и объекты, традиционные для браузеров. Преимуществом такого подхода является то, что один и тот же код может быть разделён между сервером и клиентом.

Javascript на стороне клиента, как следует из названия, – это код javascript, выполняемый на стороне клиента, типичным сценарием этого является доступ к веб-сайту и запуск кода javascript. Выполняемый код выполняется на клиентском компьютере.

## 15. Серверное программирование. Особенности, обзор технологий и языков программирования. Распространенные серверные фреймворки.

Бэкенд разработка - это по большей мере создание основной части функционала приложения или сайта, который запускается через интерфейсную часть, но происходит на стороне сервера.

Основные инструменты бэкенд-разработчика — серверные языки программирования. В целом в работе специалист использует разные инструменты:

* языки PHP, Python, Ruby, Java, C#, JavaScript, Go, JS;
* для хранения данных используются
  + Базы данных SQL (MySQL, PostgreSQL, SQLite).
  + Базы данных noSQL (MongoDB, Redis)
  + Хранилища объектов (AWS S3, Minio)
* фреймворки:
  + Express для Node.js
  + Django
  + Flask
  + Spring boot
  + CakePHP
  + ASP.NET Core
  + Laravel для PHP
  + Ruby on Rails для Ruby
  + Gin для Go

Круг задач бэкенд-разработчика выглядит так:

* разработка модели предметной области (домена);
* разработка платформы и основного функционала, то есть бизнес-логики;
* разработка разработка безопасных приложений, поддерживающих пользовательский интерфейс;
* настройка серверов (боевого, тестового и рабочего) и программ по мониторингу их состояний;
* использование системы контроля версий;
* настройка баз данных, создание моделей предметной области и взаимодействий между ними;
* настройка процессов непрерывной интеграции и поставки.

## 16. Технология взаимодействия клиентской и серверных частей веб-приложения. Протокол HTTP. Запросы и ответы.

HTTP (HyperText Transfer Protocol, «протокол передачи гипертекста») — протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов. Клиент-серверный протокол, когда один участник выступает клиентом и отправляет запросы, а сервер может только принимать и отвечать.

Клиентом может выступать браузер, приложение или другой сервер.

Структура HTTP запроса

Метод URI HTTP/Версия

Заголовок: Значение

Заголовок: Значение

ТелоЗапроса (если поддерживает метод)

Структура HTTP ответа

HTTP/Версия КодОтвета

Заголовок: Значение

Заголовок: Значение

ТелоОтвета

Коды состояния обозначают результат запроса

* 1xx Informational (Информационный) (100 Continue)
* 2xx Success (Успешно) (200 OK)
* 3xx Redirection (Перенаправление) (300 Multiple Choices)
* 4xx Client Error (Ошибка клиента) (404 Not Found)
* 5xx Server Error (Ошибка сервера) (500 Internal Server Error)

Методы

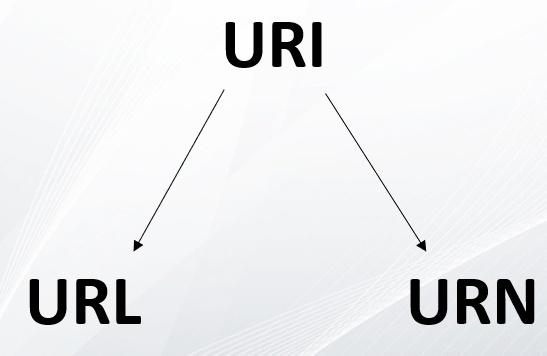
* Без тела запроса: GET, HEAD, DELETE
* С телом запроса: POST, PUT, PATCH
* Технические: OPTION, TRACE, CONNECT

При использовании архитектуры REST используются все методы, однако реализовать полноценно работающее приложение можно ограничившись только методами GET и POST.

## 17. Адресация в интернете. Основные понятия о URI, URL, URN. Структура адресов, использование.[КВ3]

Маршрутизация – поиск маршрута доставки пакета между сетями через транзитные узлы – маршрутизаторы. (это передача информации от отправителя к получателю)

URL является частью URI. *Каждый URL является URI. Каждый URN является URI. Но не каждый URI, к примеру, является URL (он может быть URN).*



**URI (Uniform Resource Identifier)** – это строка символов, которая используется для идентификации какого-либо ресурса по его адресу или по его имени, либо по тому и тому вместе. URI = URL + URN.

* **URI** = <http://ru.wikipedia.org/w/index.php?title=URI>

где:

**URL** = <http://ru.wikipedia.org/>

**URN** = w/index.php?title

**URL (Uniform Resource Locator)** – это строка символов, которая используется для идентификации какого-либо ресурса, но только по его адресу, по его местоположению.

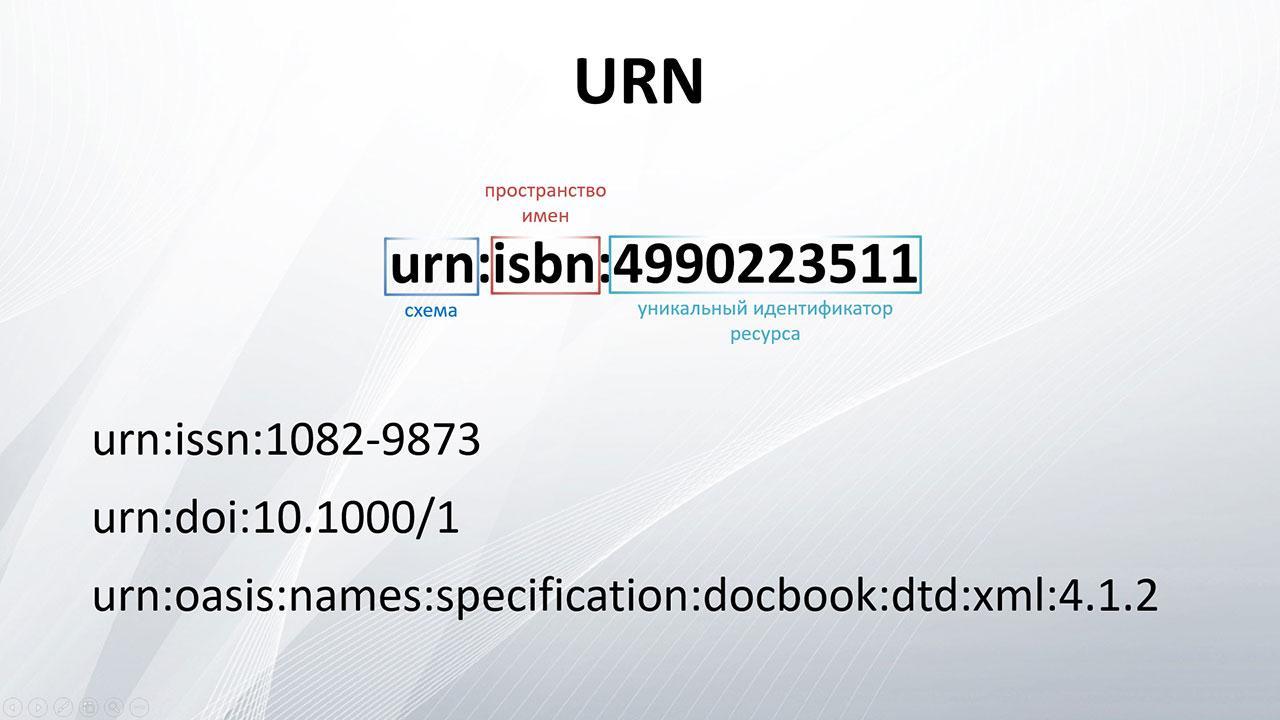
**URN (Uniform Resource Name)** – это строка символов, которая используется для идентификации какого-либо ресурса, но только по его имени.

URI может использовать и адрес, и имя при идентификации ресурса. В то время как URL и URN только адрес и только имя соответственно.

**Пример URL:** (*Любой URL состоит из нескольких компонентов. Протокол и хост являются обязательными, все остальные - нет.*)



*URN позволяет получить доступ к ресурсу по различным сетевым протоколам, обращаясь к одному и тому же имени.* в контексте веба URN практически не используется.



URN указывает неизменное имя ресурса без указания его местонахождения и способа обращения. В результате URN-имена постоянны, они не зависят от конкретных серверов и протоколов. Другими словами, URN концептуально обозначает сам ресурс, а не место, где находится какой-то ресурс (а может, уже не находится), как это делает URL.

Для нахождения ресурсов по URN-имени нужна «система разрешения URN-имён» (англ. URN resolution). Тогда человек (или программа), знающий точный URN ресурса, введёт его в систему разрешения и немедленно получит множество конкретных мест (серверов или, скажем, интернет-магазинов), где этот ресурс можно найти. В 2002 году была предложена система DDDS (англ. Dynamic Delegation Discovery System; система динамического обнаружения ресурсов), которая разрешает имена URN в URL-ссылки на конкретные местонахождения ресурсов. При этом и URN, и URL являются частью одной системы идентификации ресурсов URI.

URI,URL = http://handynotes.ru/2009/09/uri-url-urn.html

(полная идентификация файла, место(Сайт(сервер)), Каталоги, подкаталоги, имя файла, расширение)

URL,URI = http://handynotes.ru

(место(Сайт(сервер))

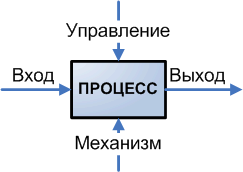
URN,URI = /2009/09/uri-url-urn.html

Название пространства имен и идентификатора в этом пространстве, иначе можно записать: %systemroot%/2009/09/uri-url-urn.html

## 18. Эволюция методологий моделирования. Моделирование деятельности и моделирование процессов. Использование методологии ARIS для моделирования деятельности предприятия. Модели, атрибуты моделей, действия над моделями, типы моделей. Объекты, свойства объектов. Связи, свойства связей. Техническая реализация методологии моделирования.[КВ4]



В 70-ых годах ХХ века при усложнении объемов и масштабов реализуемых проектов в США по заказу министерства обороны была разработана методология функционального моделирования IDEF0 (Integration Definition For Function Modeling)



Каждая из четырех сторон блока имеет строго определенное значение:

* Левая сторона обозначает входы, т. е. что поступает на вход процесса и будет преобразовано;
* Правая сторона — выход, т. е. что создается на выходе процесса в результате его выполнения;
* Верхняя сторона — управление, т. е. какие требования необходимо выполнить или чем руководствоваться при исполнении процесса;
* Нижняя сторона — механизм, т. е. какие ресурсы необходимы для исполнения процесса.

Исходя из смыслового значения сторон блока, процесс в общем виде можно определить как деятельность, которая преобразует Входы в Выходы по правилам и требованиям, установленным Управлением, с помощью определенных Механизмов.

Следующим шагом развития методологий являются 80-ые, где впервые широко стали применяться компьютеры.

Потребность в автоматизации бизнес-процессов в свою очередь вызвала необходимость их описания и формализации на принципиально другом уровне. В зависимости от целей, стоящих перед специалистами по автоматизации, было разработано достаточно большое число методологий для описания деятельности организаций, самые известные из которых:

* Функциональное моделирование с использованием диаграмм потоков данных различных нотаций (Йордана/Де Марко, Гейна-Сарсона);
* Информационное моделирование с использованием диаграмм «сущность-связь»различных нотаций (Чена, Баркера);
* Моделирование бизнес-процессов в виде цепочки событий (нотация EPC);
* Динамического функционального анализа на основе сетей Петри различного вида, функционально-стоимостного анализа.

Для автоматизации данных методологий были созданы **CASE (computer-aided software engineering)** средства, дающие возможность автоматизировать процесс описания и формализации бизнес-процессов.

Позже умные люди решили, а что если переложить CASE-средства на управление крупными компаниями, так с наличием большого опыта описания бизнес-процессов по CASE в 90-ые специалисты решили сместить акцент на решения задач менеджмента. наличие у компании модели деятельность «как есть» позволяло построить модель «как должно быть», а в дальнейшем и провести ее реструктуризацию, со значительно меньшими трудностями и рисками.

Программы, предназначенные для решения организационных вопросов управления или бизнес-моделирования, выделили в отдельный класс, который на западном рынке получил название **«BMS» (Business Modeling Software)**. К ключевым преимуществам программ этого класса можно отнести:

* Широкие возможности **визуального моделирования** бизнес-процессов компании;
* Обеспечение возможности **одноформатного**, одинакового **понимания** бизнес-процессов компании сотрудниками с разным уровнем квалификации, различных подразделений;
* Изначально встроенная поддержка **процессного подхода**;
* Представление единого информационного пространства для описания **всех** существенных **аспектов** ее **функционирования**;
* Получение различных **отчетов** из модели (например, в виде регламентных документов: Положения о подразделения, Должностные инструкции, Регламент процесса, Регламент процедуры, Матрицы ответственности и др.)

ARIS (Architecture of Integrated Information Systems) - методология и тиражируемый программный продукт для моделирования бизнес-процессов организаций.

Модели ARIS:

* Организационная схема (Organizational Chart — ОС). Отражают:
  + подразделения предприятия;
  + наименование должности и фамилии руководителей подразделений;
  + физическое местоположение отделов на предприятии.
* Функциональная модель (Function Tree — FT). Представляет собой "дерево" основных функций, реализуемых на предприятии. Модель строится иерархически — от верхнего уровня функций к нижнему (через декомпозицию).
* Процессно-событийная модель (Extended Event-Driven Process Chain — еЕРС) Предназначена для детального описания процессов, выполняемых в рамках одного подразделения, несколькими подразделениями или конкретными сотрудниками.

## 19. Разделение прав доступа и система безопасности современных многопользовательских операционных систем на примере Linux. Пользователи и группы, основные права. Основные команды для управления правами и пользователями.

Для разделения прав доступа и организации системы безопасности в Linux есть “минимальная” единица управления - *пользователь*. Каждому пользователю выдается входное имя - login, которому соответствует уникальный идентификатор пользователя *(User Identifier, UID),* с помощью которого и работает механизм прав доступа.

Пользователи объединяются в группы для более простого управления правами доступа. Как группа содержит несколько пользователей, так и пользователь может принадлежать нескольким группам. Группы также имеют свой уникальный идентификатор *(Group Identifier, GID)*

Такая организация используется для ограничения прав доступа к различным объектам ОС Linux - файлам, каталогам, устройствам. На уровне файловой системы в UNIX определяется три вида доступа: *чтение* (read, **r**), *запись* (write, **w**) и *использование* (execution, **x**):

* Право на чтение из файла дает доступ к содержащейся в нем информации
* Право записи в файл дает возможность изменять содержащуюся информацию в файле
* Право использования дает возможность запускать исполняемые файлы. Например, для исполнения bash-скрипта у текущего пользователя обязательно должно быть право на его исполнение.

Для каждого файла имеется список того, что с ним может делать владелец (если совпадает UID процесса и файла), член группы владельцев (если совпадает GID) и кто угодно (если ничего не совпадает). Рассмотрим следующий файл: *script.sh alice:users rwx r-- ---*

Из этой строки можно сделать следующие выводы:

1. Пользователь alice имеет доступ к чтению, изменению и запуску скрипта
2. Пользователи из группы users имеют доступ к чтению скрипта
3. Все остальные пользователи не имеют никаких прав

Запись *script.sh alice:users rwx r-- ---* аналогична записи *script.sh alice:users 740 (7 = 111 в двоичной, 1 - наличие права, 0 отсутствие => rwx; 4 = 100 => r--; 0 = 000 => ---)*. Таким образом, система доступа к файлу организована всего за счет 9 битов.

Тем не менее есть 3 дополнительных бита, необходимые для задания прав доступа. Здесь важно отметить, что механизм rwx для файлов и директорий трактуется по-разному:

* Право на чтения в каталог дает возможность изучить содержимое каталога (список файлов, их атрибуты)
* Право на запись дает возможность создавать и удалять файлы в этом каталоге
* Право на исполнение каталога означает право переходить в этот каталог. Если вы, как владелец, хотите дать доступ другим пользователям на просмотр какого-то файла в своем каталоге, вы должны дать им право доступа в каталог, т. е. дать им "право на выполнение каталога".

1. Sticky-bit. Впервые sticky bit появился в пятой редакции UNIX в 1974 году для использования в исполняемых файлах и применялся для уменьшения времени загрузки наиболее часто используемых программ. После закрытия программы код и данные оставались в памяти, а следующий запуск происходил быстрее.Сейчас этот бит используется в основном в общих каталогах, таких как /var или /tmp, чтобы пользователи могли создавать файлы, читать и выполнять их, но не могли удалять файлы, принадлежащие другим пользователям.
2. setuid bit - флаг прав доступа в Unix, который разрешает пользователям запускать исполняемые файлы от имени другого пользователя. Если на исполняемый файл установлен бит suid, то при выполнении эта программа автоматически меняет на идентификатор того пользователя, который является владельцем этого файла. После получения повышенных прав приложение может выполнять задачи, выполнение которых обычному пользователю недоступно.
3. setgid bit - флаг, аналогичный suid, но нацеленный на повышение прав доступа группы.

Команда для изменения владельца файла - chown. Синтаксис прост “chown username.usergroup /what/the/file”. Данная команда установит пользователя username и группу usergroup владельцами файла /what/the/file. Схожая команда chgrp, которая позволяет изменить только владельца группы.

Основной командой для работы с правами доступа является chmod

Команда может исполнять команды в абсолютном и относительном режиме:

* абсолютный: chmod 755 somefile (аналог: chmod rwxrw-rw- somefile) - в явном виде указываются все права для файла
* относительный: chmod u+x,g+w,o-r somefile - в относительном виде указываются изменения: для пользователя (u) добавляется право на исполнение (+x), для группы (g) - на запись (+w), а для всех остальных (o) - забирается право на чтение (-r)

Для добавления прав на три особых бита также используется эта команда в двух способах записи:

* chmod 2755 /somedir - 2 = 010 => 0 SUID, 1 SGID, 0 sticky bit
* chmod u+s, g+s, +t /somedir - включение всех трех специальных битов для директории /somdir

## 20. Работа в командной строке bash. Основные понятия и структура команды. Основные команды для перемещения по каталогам, просмотра файлов и информации о содержимом файлов.

**Bash** — это командная оболочка для UNIX-подобных операционных систем (UNIX, GNU/Linux, MacOS). Она дает пользователю систему команд для работы с файлами и папками, поиском, настройкой окружения и позволяет управлять ОС прямо из командной строки.

Структура:

1. На первом месте идет имя команды
2. Далее за именем команды идут опции, расширяющие или модифицирующие поведение команды
3. За ключами идут аргументы, на которые направлено действие команды

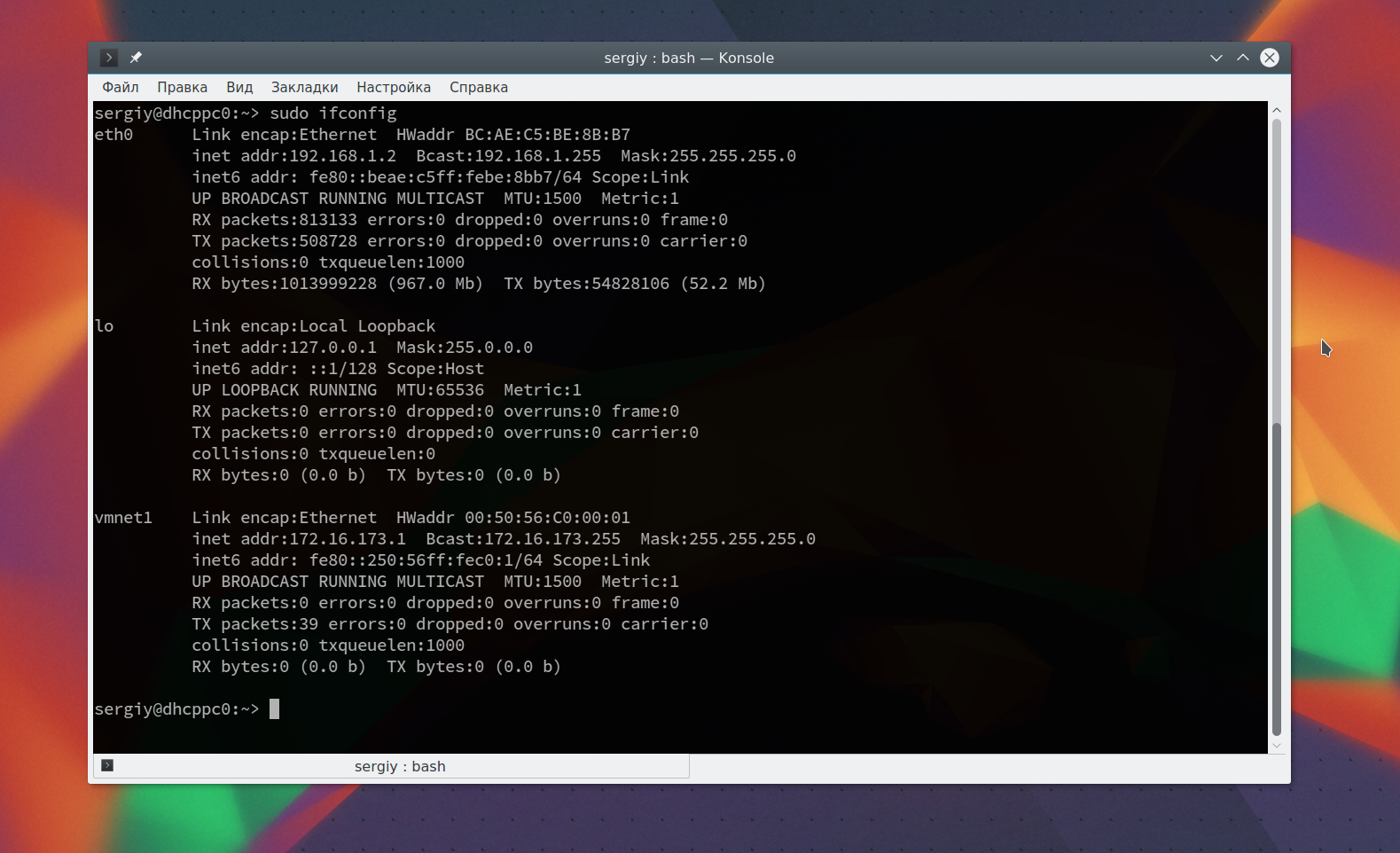
Основные команды:

* pwd. Отображает текущее местоположение в структуре каталогов.
* cd. Позволяет перейти в новый каталог
* mkdir. Создаёт новый каталог в текущем каталоге.
* man. Отображает руководства по командам.
* cat. Считывает файл, переданный как аргумент, и выводит его содержимое по стандартному каналу вывода.
* echo. Выводит свои аргументы по стандартному каналу вывода (наподобие print)
* head. Читает первые 10 строк любого переданного текста и выводит их по стандартному каналу
* tail. Работает аналогично команде head, но читает строки с конца
* ls. Выводит содержимое директории

## 21. Настройка и диагностика сетевых подключений в современных операционных системах на примере Linux. Удаленный доступ к операционной системе по протоколу SSH.

**Настройка сетевого подключения в Linux происходит при помощи команды ifconfig.**

Если ввести команду ifconfig без опций и параметров, будут показаны все сетевые интерфейсы пк.



**Подключение и отключение сетевого интерфейса происходит при помощи команд:**

ifconfig \*название интерфейса\* up для включения

ifconfig \*название интерфейса\* down для выключения

**Настройка сети происходит в несколько команд:**

ifconfig \*название интерфейса\* up - включаем

ifconfig \*название интерфейса\* add 192.168.1.10 устанавливаем ip-адрес

ifconfig \*название интерфейса\* netmask 255.255.255.0 устанавливаем маску сети

ifconfig \*название интерфейса\* broadcast 192.168.1.255 устанавливаем шлюз для выхода в сеть

route add default gw 192.168.1.1 устанавливаем DNS сервер

Если у нас динамический ip, то также выполняем команду

ifconfig eth0 0.0.0.0 0.0.0.0 && dhclient

**Что такое SSH?**

На большинстве серверов используется Linux, который не имеет графического интерфейса, ради экономии ресурсов сервера. Более того, командная строка предоставляет больше возможностей чем графический интерфейс. Грубо говоря, подключение по SSH, это когда вы подключаетесь к удаленному серверу через командную строку на вашем компьютере и через эту же строку можете выполнять команды и настраивать сервер.

**Подключение к серверу происходит при помощи команды ssh:**

ssh имя\_пользователя@айпи\_адрес

чаще всего это будет что-то вроде этого:

ssh admin@192.168.1.2

если ваш сервер имеет нестандартный порт, то вы можете указать его при помощи параметра -p:

ssh admin@192.168.1.2 -p 2223

Если такой сервер существует и вы успешно подключились к нему, то в командной строке появится сообщение с просьбой ввести пароль. Если вы ввели верный пароль, то вы войдете на сервер под своей учетной записью и сможете выполнять команды.

## 22. Основные сетевые службы. Веб-технологии - необходимое программное обеспечение, назначение схема работы, основные протоколы.

## **Основные сетевые службы**

Сетевой службой называют совокупность серверной и клиентской частей ОС, предоставляющих доступ к конкретному типу

* почта (SMTP) - сетевой протокол, предназначенный для передачи электронной почты.
* передача файлов (FTP) - протокол передачи файлов по сети.
* файловые серверы (NFS) - протокол сетевого доступа к файловым системам.
* cлужба времени (NTP) - протокол взаимодействия с серверами точного времени
* удаленный терминал и выполнение (SSH) - то сетевой протокол, используемый для подключения к удаленным компьютерам и управлениями ими с помощью технологии туннелирования.
* удаленное использование рабочей среды X (XDMCP – X Display Manager Control Protocol) - протокол, обеспечивающий удаленное подключение графического терминалов к графическому серверу. Аналог RDP в Windows.
* Веб (HTTP, HTTPS) - Протокол передачи данных, html, json и другое.

## **Веб-технологии - необходимое программное обеспечение, назначение схема работы, основные протоколы.**

Под веб-технологиями обычно подразумевается весь стек технологий обеспечивающий взаимодействие пользователя с веб-сервисом. Пример: Браузер -> DNS-сервер -> Сервер нужного ресурса -> БД сервера и тд.

Разберем весь стек:

**Веб-клиент** - программное средство работающее на компьютере пользователя и обрабатывающее запросы \ ответы веб-сервера. Например: Браузер, мобильное приложение и тд.

**Брандмауэр** - программное средство, контролирующее сетевую активность на основе набора правил. Основная задача - блокировка несанкционированных соединений и передачи несанкционированных данных. Например MS Firewall на Windows.

**DNS-сервер** - транслирует доменное имя ресурса в ip-адрес. Например нам нужна главная страница сайта фу, у нас есть доменное имя сайта, но это доменное имя ничего не говорит компьютеру, ему нужен ip, тогда компьютер обращается к сертифицированным DNS-серверам, те по домену выдают ему валидный ip, подключившись по которому, мы получим главную страницу.

**Серверные ОС** - предназначены для выполнения серверного ПО. Так-то мы и на домашнем ПК можем поставить сервер, но зачем, когда есть специальные ОС. Например: CentOS, Ubuntu, MS Windows Server

**Веб-приложение** - широкое понятие серверного приложения, как правило состоит из 3 основных компонентов, их может быть больше или меньше:

1. *Веб-сервер* - программы-обертки для бэкенда или посредники для улучшения коммуникаций между клиентом и нашей реализацией логики веб-приложения. К таким программам относятся Apache или Nginx.
2. *Веб-Фреймворк* (Django, Flask, Express) или *чистая реализация* (Python, Node.js) - в них мы пишем всю логику веб-приложения, факт только в том, что фреймворки удобнее, чем реализовывать все руками.
3. *СУБД* - хранилище данных и их управление. Например PostgreSQL или MongoDB

Такой стек обеспечивает полное взаимодействие клиента и сервера в рабочих условиях.

Краткая схема работы такая: Мы хотим открыть сайт, зная его домен, через браузер (**веб-клиент**), запрос проходит через **Брандмауэр**, отправляется к **DNS-серверу** для получения IP, по полученному IP обращаемся к **веб-приложению**, которое стоит на какой-то **Серверной** ОС, это **веб-приложение** генерирует ответ (задействуя либо веб-сервер, либо все приложение), он идет к нам, проходит опять через **Брандмауэр** и отображается в браузере (**веб-клиенте**)

Подробная схема работы: <https://svyatoslav.biz/wp-pics/webtech_01_web_technology_ru.png>

Протоколы:

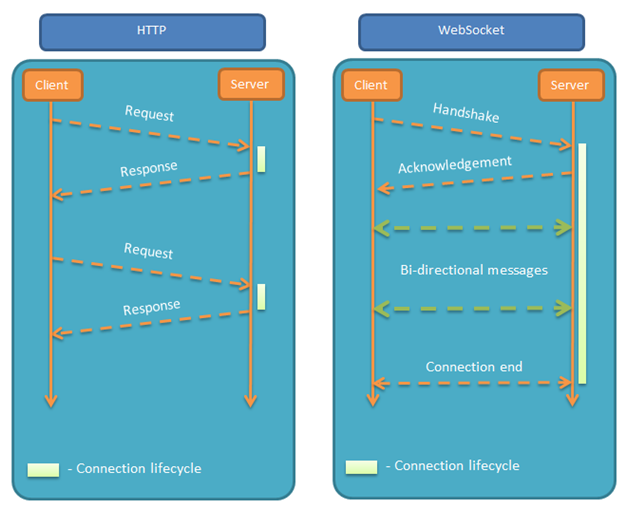
Многие протоколы уже были указаны в начале ответа, так что приведу только те, что чаще всего встречаются во взаимодействие с современными веб-сервисами:

Прикладные:

**HTTP** – протокол прикладного уровня передачи данных, используется для передачи произвольных данных

**HTTPS** – безопасный протокол передачи гипертекста- расширение протокола HTTP, поддерживающее шифрование посредством криптографических протоколов SSL и TLS

**WebSocket** — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером через постоянное соединение. Данные передаются по нему в обоих направлениях в виде «пакетов», без разрыва соединения и дополнительных HTTP-запросов.



Транспортные:

* **TCP** (Transmission Control Protocol, протокол управления передачей) Предназначен для управления передачей данных интернета. Пакеты в TCP называются сегментами.

Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. TCP осуществляет надежную передачу потока байтов от одного процесса к другому. TCP реализует управление потоком, управление перегрузкой, рукопожатие, надежную передачу.

* **UDP** (User Datagram Protocol — протокол пользовательских датаграмм)

С UDP компьютерные приложения могут посылать сообщения другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных.

UDP использует простую модель передачи, без явных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных.

Датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа, но гарантируется, что если они придут, то в целостном состоянии.

UDP подразумевает, что проверка ошибок и исправление либо не нужны, либо должны исполняться в приложении. Чувствительные ко времени приложения часто используют UDP, так как предпочтительнее сбросить пакеты, чем ждать задержавшиеся пакеты, что может оказаться невозможным в системах реального времени. При необходимости исправления ошибок на сетевом уровне интерфейса приложение может задействовать TCP

## 23. Общая характеристика языка программирования C#. Схема выполнения, синтаксис, система типов.

Объектно-ориентированный язык с C-подобным синтаксисом и статической типизацией.

Компилируется в байт-код и выполняется в среде CLR (основной реализацией которой является среда .NET)



Система типов

Все типы можно разделить на четыре категории:

1. Типы-значения ( value ), или значимые типы.
2. Ссылочные ( reference ).
3. Указатели ( pointer ).
4. Тип void.

| **Логический тип** | | |
| --- | --- | --- |
| **Имя типа** | **Значения** | **Размер** |
| bool | true, false | 8 бит |
| **Арифметические целочисленные типы** | | |
| **Имя типа** | **Диапазон** | **Размер** |
| sbyte | -128 — 127 | Знаковое, 8 Бит |
| byte | 0 — 255 | Беззнаковое, 8 Бит |
| short | -32768 —32767 | Знаковое, 16 Бит |
| ushort | 0 — 65535 | Беззнаковое, 16 Бит |
| int | \approx (-2\*10^9 — 2\*10^9) | Знаковое, 32 Бит |
| uint | \approx (0 — 4\*10^9) | Беззнаковое, 32 Бит |
| long | \approx (-9\*10^18 — 9\*10^18) | Знаковое, 64 Бит |
| ulong | \approx (0— 18\*10^18) | Беззнаковое, 64 Бит |
| **Арифметический тип с плавающей точкой** | | |
| **Имя типа** | **Диапазон** | **Точность** |
| float | +1.5\*10^-45 -/+3.4\*10^38 | 7 цифр |
| double | +5.0\*10^-324 -/+1.7\*10^308 | 15-16 цифр |
| **Арифметический тип с фиксированной точкой** | | |
| **Имя типа** | **Диапазон** | **Точность** |
| decimal | +1.0\*10^-28 - +7.9\*10^28 | 28-29 значащих цифр |
| **Символьные типы** | | |
| **Имя типа** | **Диапазон** | **Точность** |
| char | U+0000 - U+ffff | 16 бит Unicode символ |
| string | Строка из символов Unicode |
| **Объектный тип** | | |
| **Имя типа** | **Примечание** | |
| object | Прародитель всех встроенных и пользовательских типов | |

Функция System.Type GetType () - возвращает системный тип текущего объекта

## 24. Объектно-ориентированное программирование в C#. Принципы, особенности реализации. Синтаксис.

Есть три основных принципа объектно-ориентированного программирования:

* Инкапсуляция

Предназначена для изоляции контрактных обязательств абстракции (протокол/интерфейс) от их реализации. На практике это означает, что класс должен состоять из двух частей: интерфейса и реализации. В реализации большинства языков программирования, обеспечивает механизм сокрытия, позволяющий разграничивать доступ к различным частям компонента.

Пример: У класса ImageInfo есть закрытые данные Width и Height это инкапсуляция. У класса ImageInfo есть открытые методы GetWidth, GetHeight мы их и используем.

// мой класс

class ImageInfo

{

// инкапсулированные данные (скрытые модификатором доступа protected)

protected int Width;

protected int Height;

// открытый конструктор

public ImageInfo(string filename)

{

// load image

Image image = Image.FromFile(filename); // класс из библиотеки C# System.Drawing

// get image size

Width = image.Width;

Height = image.Height;

}

// открытые методы

public int GetWidth()

{

return Width;

}

public int GetHeight()

{

return Height;

}

}

// главный класс программы

class Program

{

static void Main(string[] args)

{

// создаем объект

ImageInfo img = new ImageInfo("D:\Bear.jpg");

// получаем значение

int w = img.GetWidth();

int h = img.GetHeight();

}

}

* Наследование

Позволяет описать новый класс (класс-наследник) на основании старого (родительского). Наследник перенимает методы и свойства класса-родителя.

class Animal

{

...

}

class Dog : Animal // наследование

{

...

}

* Полиморфизм

(Кароч вкратце - Это способность объекта в процессе своего выполнения ссылаться на экземпляры собственного класса или любого класса-потомка. Классы-потомки могут переопределять метод.

class Person {

constructor(name) { this.name = name; }

me() { return `My name is ${this.name}`; }

}

const axel = new Person('Axel');

console.log(axel.me()); // -> 'My name is Axel'

class Employee extends Person {

constructor (name, salary) {

super(name);

this.salary = salary;

}

me() {

return `My name is ${this.name} and my salary is ${this.salary}`;

}

}

const nick = new Employee('Nick', 3000);

console.log(nick.me()); // -> 'My name is Nick and my salary is 3000'

)

Свойство системы использовать объекты с одинаковым интерфейсом без информации о типе объекта и организации объекта. Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе наследования.

Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций».

Полиморфизм — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с абстракцией, инкапсуляцией и наследованием).

Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному — интерфейс, класс. Общность имеет внешнее и внутреннее выражение:

* внешняя общность проявляется как одинаковый набор методов с одинаковыми именами и сигнатурами (именем методов и типами аргументов и их количеством);
* внутренняя общность — одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы. Возможность приписывать разную функциональность одному методу (функции, операции) называется перегрузкой метода (перегрузкой функций, перегрузкой операций).

// интерфейс IGraphic

interface IGraphic

{

void ShowText(string text);

}

// класс ConsoleApp

class ConsoleApp : IGraphic

{

public void ShowText(string text)

{

Console.WriteLine("Рисуем текст " + text + " в консольном приложении");

}

}

// класс Phone

class Phone : IGraphic

{

public void ShowText(string text)

{

// рисуем текст на телефоне

}

}

class Program

{

static void Main(string[] args)

{

IGraphic graphic = null;

// создаем IGraphic реализация ConsoleApp

graphic = new ConsoleApp();

// Это полиморфизм мы вызываем метод ShowText у интерфейса IGraphic

// В реальности вызовется ShowText у класса кто реализовал интерфейс IGraphic

graphic.ShowText("Hello");

// создаем IGraphic реализация Phone

graphic = new Phone();

// Это полиморфизм мы вызываем метод ShowText у интерфейса IGraphic

// В реальности вызовется ShowText у класса кто реализовал интерфейс IGraphic

graphic.ShowText("Hello");

}

}

Ещё есть абстракция - использование только тех характеристик объекта, которые с достаточной точностью представляют его в данной системе. Основная идея состоит в том, чтобы представить объект минимальным набором полей и методов и при этом с достаточной точностью для решаемой задачи. Но ее часто не включают в этот перечень (примера на C# не будет потому что любой класс, который создается - это уже пример абстракции)

## 25. Фреймворк .NET. Назначение, основные понятия, применение. Объектно-реляционное отображение (ORM).

Платформа .NET Framework — это технология, которая поддерживает создание и выполнение веб-служб и приложений Windows. Платформа .NET Framework состоит из общеязыковой среды выполнения (среды CLR) и библиотеки классов .NET Framework.

Немаловажное свойство .NET в том, что программу или библиотеку, написанную на любом языке с поддержкой CLR можно использовать потом в любом другом языке с поддержкой CLR. (Например часто встречается C#+F# в одном проекте).

Слово Framework в названии обозначает, что .NET абстрагирует программиста от компьютерного железа. Программист под .NET взаимодействует с .NET, а вся работа по оптимизациям, управлению памятью, вообще совместимостью лежит на фреймворке.

Кстати, запуск программы на разных компьютерах и даже в разных операционных системах - тоже одна из задач, которую .NET решает.

Основой платформы .NET Framework является среда CLR. Среду выполнения можно считать агентом, который управляет кодом во время выполнения и предоставляет основные службы, такие как управление памятью, управление потоками и удаленное взаимодействие. При этом средой накладываются условия строгой типизации и другие виды проверки точности кода, обеспечивающие безопасность и надежность. Фактически основной задачей среды выполнения является управление кодом. Код, который обращается к среде выполнения, называют управляемым кодом, а код, который не обращается к среде выполнения, называют неуправляемым кодом. Библиотека классов является комплексной объектно-ориентированной коллекцией повторно используемых типов, которые применяются для разработки приложений — начиная с обычных приложений, запускаемых из командной строки, и приложений с графическим интерфейсом (GUI) и заканчивая приложениями, использующими последние технологические возможности ASP.NET, такие как веб-формы и веб-службы XML.

ORM — это технология программирования, которая создает слой между реляционными базами данных и объектно-ориентированными языками программирования без необходимости написания SQL-запросов.

### **Как работает ОRМ?**

ORM выстраивает модели объектно-ориентированной программы с высоким уровнем абстракции. Другими словами, она создает уровень логики без основных деталей кода. Отображение описывает отношения между объектом и данными, не зная, как эти данные структурированы. Далее модель можно использовать для подключения приложения к коду SQL, который необходим для управления операциями с данными. Этот «сантехнический» тип кода не нужно переписывать, что значительно экономит время разработки.

# **1.3. Вопросы на основе содержания дисциплин профиля «ИТ-сервисы и технологии обработки данных в экономике и финансах»**

## 1. Понятие, классификация и принципы построения банковских информационных систем (БИС).

**Информационная система** — это взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации для достижения цели управления. ( система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы, которые обеспечивают и распространяют информацию.)

**БИС** – программно-технологический комплекс, охватывающий совокупность взаимосвязанных автоматизированных банковских операций и задач.

Учет и контроль в банке представлены операционным и бухгалтерским учетом, которые тесно связаны между собой (каждый лицевой счет находится под определенным балансовым). Статистический учет позволяет собрать сведения об изменении отдельных покажзателей за длительный период.

Рост требований к современным банковским системам определяется рядом факторов: изменение экономической обстановки в России. Основные факторы влияния:

* факторы внешней среды (действия конкурентов, внешние ресурсы);
* факторы внутренней среды (история банка, структура банка);
* факторы макросреды (политические, социально-экономические, природные).

*Принципы построения БИС:*

* комплексный подход;
* модульный принцип построения;
* открытость технологий;
* гибкость настройки модулей банковской системы;
* масштабируемость;
* многопользовательский доступ к данным;
* моделирование банка и его бизнес-процессов.

Основные требования, предъявляемые к интегрированной автоматизированной банковской системе:

1. Открытость интегрированных АБС;

2. Соответствие основным банковским принципам;

3. Обеспечение единого информационного пространства.

4. Настраиваемость на конкретные приложения и пользователей;

5. Обеспечение управляемости банковской деятельности;

6. Надежность, защищенность и безопасность;

7. Дисциплина и единый регламент документирования, сопровождения и модификации.

Структуризация (построение) ИБС предусматривает выделение элементов по определенным принципам.

*1) Функциональный критерий структуризации ИБС.*

Выделяются функциональные модули ИБС –

* расчетно-кассовое обслуживание,
* учет коммерческих кредитов,
* учет депозитов,
* обслуживание физических лиц,
* операции с ценными бумагами,
* операции на денежных рынках,
* вексельный учет,
* хозяйственные договора и др.

Структура ИБС представляется как совокупность функциональных модулей, построенных в едином технологическом ключе, объединенных вокруг единого финансового ядра и работающих на единой аппаратно-программной платформе.

*2) Модульный принцип построения ИБС.*

Предполагает выделение элементов (блоков, информа-ционных модулей) ИБС, набор которых может варьироваться, отражая специфику, потребности банка и его масштаб. Функционально связанные модули образуют единую ИБС. На практике чаще всего используется функциональное разделение, что позволяет пользователю связать отдельные модули в единую информационную систему, максимально отражающую специфику и потребности каждого банка.

*По функциональному принципу выделяют модули:*

* операционный день банка (банковский учет);
* расчетно-кассовое обслуживание (РКО);
* кредитование; депозитарий и др.

*По объектному принципу выделяют модули:*

* головного банка;
* филиала;
* отделения;
* представительства.

*3)**Принцип единого информационного пространства*

На уровне ИБС единое информационное пространство можно интерпретировать как возможность системы оперировать любыми данными, формирующимися в процессе функционирования системы, при этом важное значение имеет и принцип открытости технологий, способных взаимодействовать с различными внешними системами, финансовыми и информационными структурами.

К внешней среде, с которой взаимодействует коммерческий банк относятся:

* обменные пункты,
* участники электронного денежного обращения с использованием пластиковых карточек,
* клиенты,
* филиалы банка,
* зарубежные банки.

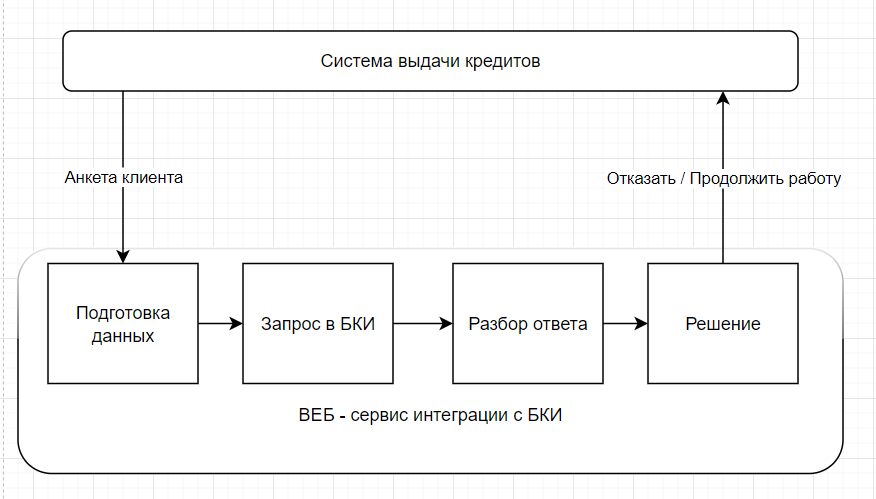
Элементы современной ИБС работают на единой информационной платформе, в основе которой лежит универсальное финансовое ядро. Результаты одного модуля могут менять информационную среду другого модуля

## 2. Реализация решения задач кредитного модуля банковских информационных систем.

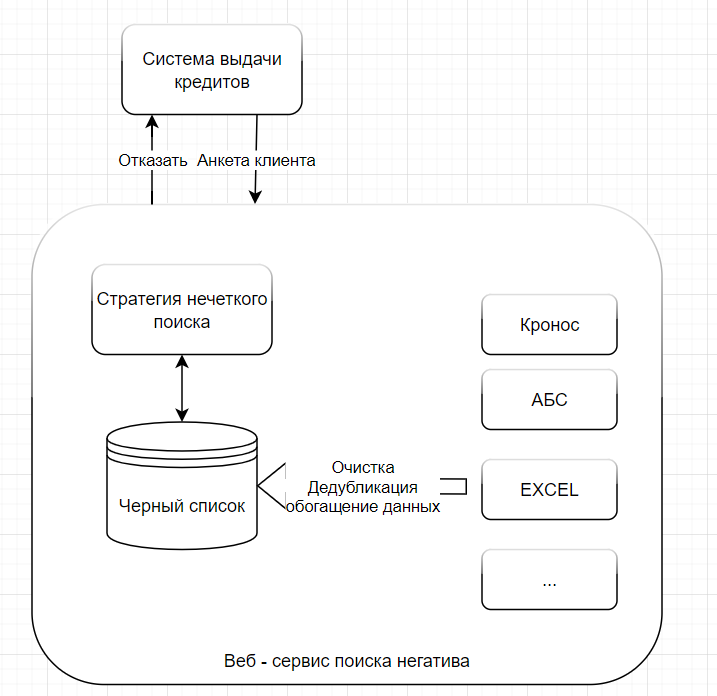
Модуль учета коммерческих кредитов имеет своей задачей обеспечение регистрации кредитных заявок клиентов, анализа кредитоспособности заемщиков, учета заключенных кредитных договоров различных видов, в том числе кредитных линий, овердрафтов, договоров ипотечного и потребительского кредитования. Кроме того, модуль используется для бухгалтерского отражения кредитных операций с автоматическим формированием платежных, внутренних и внебалансовых документов, формирования внутренних документов различного вида и бухгалтерской отчетности в соответствии с требованиями Банка России.

Реализация: БКИ - бюро кредитных историй

Работа с БКИ



Поиск в черных списках



## 3. Применение технологии Big Data в банковской сфере

**Big Data или большие данные** — это структурированные или неструктурированные массивы данных большого объема. Их обрабатывают при помощи специальных автоматизированных инструментов, чтобы использовать для статистики, анализа, прогнозов и принятия решений.

На сегодняшний день данную технологию применяют крупнейшие российские банки такие как ПАО «Сбербанк», ПАО «ВТБ», АО «Альфа-Банк», АО «Россельхозбанк» и другие (всего около тридцати).

Уже сегодня Big data имеет положительный результат в банковской сфере. К примеру, еще некоторое время назад, требовалось несколько дней, чтобы рассмотреть информацию лица, подавшего заявление на получение кредитной карты. А сейчас данная процедура занимает несколько минут.

Также банки используют большие данные для анализа поведения клиентов для того, чтобы узнать об их поведение на рынке, узнать об их предпочтениях, об отношении к введенным новым банковским услугам, и для того, чтобы определить какую услугу им предложить.

**Задачи, решаемые технологией Big Data**

· сбор и анализ информации об аномалиях во внутренней сети в режиме, приближенном к реальному времени, на основе технических данных оборудования канального и сетевого уровня модели OSI;

· противодействие мошенничеству в отношении держателей пластиковых банковских карт;

· контроль и оценка эффективности работы системы обеспечения информационной безопасности с использованием опробованных в смежных организациях аналитических моделей;

· категоризация работников в соответствии с моделью внутреннего нарушителя и динамическое перераспределение на основе дополнительно полученной информации;

· оценка рисков информационной безопасности (автоматическое заполнение скоринговых карт по методикам аудиторов или риск-менеджеров).

**Сдерживающие факторы развития технологии**

Основными факторами, препятствующими массовому применению технологий Big Data в отечественных компаниях, являются:

1. Высокая стоимость владения и эксплуатации инфраструктуры Big Datа на каждом из этапов, в том числе:

· аналитическая обработка;

· нормализация/фильтрация;

· хранение/предоставление доступа;

· формализация нечетких задач;

· создание и корректировка моделей;

· исследование неявных зависимостей;

· визуализация результатов;

· прогнозирование изменений в динамике;

2. Сложность технической реализации процесса гарантированного хранения информации из внешних и внутренних источников на протяжении долгого времени.

3. Критичные требования по производительности и отказоустойчивости источников, агрегаторов, серверов приложений, аналитических и графических модулей и пр.

4. Нечеткая постановка задачи и негарантированность результатов от применения технологий Big Data для заказчика.

5. Высокая персональная зависимость, поскольку квалифицированных аналитиков с техническим опытом в настоящее время на HR-рынке найти достаточно сложно и еще сложнее удержать, а развитие аналитических моделей, созданных другими, сравнимо по трудоемкости с повторным созданием модели с нуля.

Бл6. Комплаенс-риски в контексте территориальных и трансграничных вопросов передачи и хранения данных.

Поскольку большинство банковских клиент-ориентированных систем призваны обеспечить высокий уровень доступности сервиса для клиентов (работают в режиме реального времени), размер файлов с технической информацией на несколько порядков превышает объемы хранимой бизнес-информации

## 4. Системы дистанционного банковского обслуживания.

Дистанционное банковское обслуживание (ДБО) позволяет физическим или юридическим лицам **удаленно совершать расчеты и другие банковские операции по расчетному счету**.

С точки зрения используемых технологий можно выделить следующие виды ДБО:

Классический «Банк-Клиент»(толстый клиент, remote banking, home banking) - на компьютере пользователя устанавливается отдельная программа клиент, которая хранит тут же все свои данные (выписки по счетам, платежные документы). Взаимодействие с банком может осуществляться по различным каналам связи (телефонные коммутируемые или выделенные линии, через сеть Интернет).

Интернет-банкинг (интернет-клиент, тонкий клиент, On-line banking, Internet banking, WEB-banking) - это система дистанционного банковского обслуживания, работающая через обычный Интернет-браузер. С ее помощью можно осуществлять все те же действия, что и через традиционные системы, с тем отличием, что не требуется установка дистрибутива системы на компьютер пользователя. Подробнее см. Интернет-банкинг.

Мобильный банкинг(телефонный банкинг, SMS-banking) - оказание услуг ДБО с использованием телефонной связи. Как правило, такие системы имеют ограниченный набор функций. Чаще всего это информационный сервис (выписки об операциях по счетам, остаток на счете и т.п.).Но некоторые банки позволяют клиентам производить различные платежи и осуществлять денежные переводы с помощью мобильного телефона.

Внешние сервисы - технологии ДБО с использованием устройств банковского самообслуживания (банкоматов, платежных терминалов, информационных киосков).

## 5. Информационные системы бухгалтерского учета (ИС БУ), их структура и классификация.

Бухучет представляет собой упорядоченную систему сбора, регистрации и обобщения информации о состоянии имущества, обязательствах и капитале организации в денежном выражении и их изменениях путём сплошного и непрерывного документального отражения всех хозяйственных операций.

Основной задачей бухучета является формирование полной и достоверной информации (бухгалтерской отчетности) о деятельности организации и ее имущественном положении.

Классификационные признаки

* Предназначение
  + для малых предприятий
  + средних предприятий
  + крупных предприятий
* Степень охвата функций бухгалтерии
  + мини бухгалтерия;
  + интегрированная система бухгалтерского учета;
  + комплексы бухгалтерских АРМ
* Принцип архитектуры технического обеспечения
  + локальные АРМ
  + ИС с сетевой архитектурой обработки информации

## 6. Подходы к классификации бухгалтерских информационных систем, их и различия.

**Автоматизированная информационная система бухгалтерского учета (АИС-БУ)** – это система, в которой информационный процесс бухгалтерского учета автоматизирован за счет применения специальных методов обработки данных, использующих комплекс вычислительных, коммуникационных и других технических средств, в целях получения и доставки информации, необходимой специалистам-бухгалтерам для выполнения функций управленческого и финансового учета.

**Структура АИС БУ** (сплошная линия – прямая связь, пунктир – обратная связь)



**Классификация АИС БУ**

1. Степень охвата учетных функций – этот признак характеризует состав учетных задач, решаемых системой.

* Системы автоматизации отдельных разделов бухгалтерского учета
* Системы частичной автоматизации бухгалтерского учета
* Системы комплексной автоматизации бухгалтерского учета
* Системы автоматизации бухгалтерского учета, интегрированные с функциями оперативного учета
* Системы автоматизации бухгалтерского учета корпоративных информационных систем управления экономическим объектом

1. Концепция построения информационной модели учета

* Системы с унифицированной моделью учетных данных
* Системы со специализированными моделями учетных данных

1. Способ построения программной системы

* Системы, построенные на основе единого программного ядра
* Комплексы специализированных программных модулей

1. Возможность расширения состава базовых функций системы

* Системы с замкнутой функциональностью
* Системы с ограниченной развиваемой функциональностью
* Системы полностью реконфигурируемые

1. Порядок распространения и тиражирования

* Системы массово тиражируемые
* Системы малотиражные
* Системы индивидуального назначения

1. Программно-техническая платформа системы

## 7. Модель системы счетов как основа любой бухгалтерской информационной системы. Модель организации синтетического учета. Модели организации аналитического учета.[КВ2]

**Модель системы счетов** – это порядок построения и интерпретации всех синтетических и аналитических счетов, применяемых пользователем и обрабатываемых программой.

Она включает систему синтетических счетов, систему аналитических счетов и связи между ними. Модели системы счетов, реализуемые в программном обеспечении АИС-БУ (автоматизированных информационных систем бухгалтерского учета), различны. Они проявляются в построении систем синтетических и аналитических счетов, способах организации связей между ними, а также в порядке интерпретации счетов при их обработке.

**Синтетический учет** - это учет обобщенных данных бухгалтерского учета о видах имущества, обязательств и хозяйственных операций по определенным экономическим признакам, который ведется на синтетических счетах бухгалтерского учета (ФЗ).

Синтетический учет ведется на основе синтетических счетов (план счетов) бухгалтерского учета. Т.е. центральным элементом модели организации синтетического учета является компьютерный план счетов.

Компьютерный план счетов – это справочник, организованный в виде списка. Каждая запись списка соответствует определенному счету/субсчету. Компьютерный план счетов включает их характеристики, определяющие алгоритмы и технологические особенности последующей обработки счетов.

Наиболее типичными характеристиками счетов/субсчетов являются:

* код и наименование счета/субсчета;
* тип счета по отношению к балансу;
* тип сальдо счета/субсчета;
* признак наличия аналитического учета по счету/субсчету;
* признак необходимости ведения натурального (количественного) учета объектов аналитического учета, привязанных к данному счету/субсчету;
* признак необходимости ведения учета в инвалютном выражении;

увязка счетов с типовыми формами выходной информации.

**Основные модели аналитического учета**

1) **Простая модель** аналитического учета предполагает одноуровневую организацию объектов бухгалтерского учета. При этом предполагается выделение видов объектов аналитического учета (типов) (организации, отчетные лица, сотрудники и т.д.). По каждому типу открывается справочник объектов аналитического учета. Затем, через тип устанавливается связь между синтетическими и аналитическими счетами. При этом синтетическому счету может соответствовать только один аналитический счет. Для аналитического счета в структуре записи операции отводится отдельное поле. При формировании проводки программа анализирует введенный пользователем код синтетического счета и через тип аналитического счета предлагает ввести объекты аналитики из справочника.

2) **Иерархическая модель** организации аналитического учета. Суть состоит в том, что признаки, по которым классифицируются объекты аналитического учета, выстраиваются по иерархии. Следовательно, множество объектов аналитического учета делятся на соподчиненные объекты бухгалтерского учета, таким образом, выделяют аналитические счета энных уровней. И модель получается в виде графа Дерева. В отличие от простой модели на этапе формирования проводки здесь подключается несколько справочников аналитического учета, выстроенных в определенной иерархии.

## 8. Понятие базы данных. Виды баз данных. Проектирование баз данных.

**База данных** – поименованная и организованная совокупность взаимосвязанных данных, которые отражают состояние объектов конкретной предметной области.

Основные задачи проектирования БД:

* Обеспечение хранения в БД всей необходимой информации
* Обеспечение возможности получения данных по всем необходимым запросам
* Сокращение избыточности и дублирования и обеспечение целостности БД

По типу хранимой информации БД делятся на:

* документальные,

Базы данных, объединяющие документы, сгруппированные (организованные) по разным свойствам

* фактографические

объединяют данные по факту совершения события (дата выпуска товара, год рождения сотрудника).

* лексикографические.

объединяют словари, классификаторы, и т.л. документы.

По обращению к ним:

* локальная

предназначенная для локального использования одним пользователем.

* общие(интегрированные, централизованные)

предоставляют коллективный доступ к данным

* Распределительные

аналогичны интегрированным, но могут быть физически разнесены на разные машины, и при этом логически считаться единым целым.

По способу организации данных:

* иерархические

в виде [древовидной](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B5%D0%B2%D0%BE%D0%B2%D0%B8%D0%B4%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0) ([иерархической](https://ru.wikipedia.org/wiki/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%8F)) структуры, состоящей из объектов ([данных](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5)) различных уровней.

* сетевые

логическая [модель данных](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), являющаяся расширением [иерархического подхода](https://ru.wikipedia.org/wiki/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных.

Разница между [иерархической моделью данных](https://ru.wikipedia.org/wiki/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и сетевой состоит в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков.

* реляционные

самая используемая модель БД. Эта модель используется везде, где есть формализованная информация. Основа этой модели таблица, а взаимоотношения данных происходят по «доменам», «атрибутам», «кортежам»

* мультимодельные

совмещают в себе несколько подходов

**Этапы проектирования БД:**

1. Исследование предметной области

Предметная область - это часть реального мира, данные о которой мы хотим отразить в БД.

В теории проектирования информационных систем предметную область принято рассматривать в виде трех представлений:

* представление предметной области в том виде, как она реально существует
* как ее воспринимает человек(проектировщик БД)
* как она может быть описана с помощью символов.

Т.е. говорят, что мы имеем дело с реальностью, описание реальности и с данными, которые отражают представление.

1. Создание инфологической модели

Концептуальное (инфологическое) проектирование - построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции.

**Инфологическая модель** (информационно-логическая модель) — ориентированная на человека и не зависимая от типа СУБД модель предметной области, определяющая совокупности информационных объектов, их атрибутов и отношений между объектами, динамику изменений предметной области, а также характер информационных потребностей пользователей.

ЦЕЛЬ: обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты).

Конкретный вид и содержание концептуальной модели базы данных определяется выбранным для этого формальным аппаратом. Обычно используются графические нотации, подобные ER-диаграммам. Как правило концептуальная модель базы данных включает в себя:

• описание информационных объектов или понятий предметной области и связей между ними.

• описание ограничений целостности, т.е. требований к допустимым значениям данных и к связям между ними.

## 9. Общая характеристика языка запросов SQL. Декларативная парадигма. Основные операторы, виды операторов.

**SQL** — формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей системой управления базами данных (СУБД). SQL основывается на исчислении кортежей.

**Язык SQL представляет собой совокупность**

· операторов,

· инструкций,

· и вычисляемых функций.

SQL является прежде всего информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. SQL можно назвать языком программирования, при этом он не является тьюринг-полным, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.

**Декларативность языка**

Декларативность языка заключается в том, что, в отличие от обычных языков, в которых программный код формирует последовательность действий, которые должен выполнить компьютер, в декларативных действиях код представляет из себя задание, то есть, практически, описание результата его выполнения. Соответственно, при составлении SQL-запроса, программист не решает задачу по работе с базой данных, а составляет описание её результата, а каким образом данная задача будет выполнена – это решает СУБД, анализируя и выполняя запрос.

**Операторы**

SQL содержит 4 группы операторов:

* Data Definition Language (DDL). CREATE, DROP, ALTER и др.
* Data Manipulation Language (DML). INSERT, DELETE, SELECT, UPDATE и др.
* Data Control Language (DCL). GRANT / REVOKE , LOCK / UNLOCK , SET LOCK MODE
* Transaction Control Language (TCL). COMMIT, ROLLBACK и др

## 10. Оператор SELECT в SQL. Общий синтаксис, наиболее распространенные блоки. Условия, сортировки, ограничения. Примеры.

Оператор языка SQL SELECT - основная строительная конструкция для создания любого, простого или сложного запроса к базе данных.

Запрос с оператором SELECT для выбора всех столбцов таблицы имеет следующий синтаксис:

SELECT \* FROM ИМЯ\_ТАБЛИЦЫ

Для выбора определённых столбцов таблицы нам потребуется вместо звёздочки перечислить через запятую названия всех столбцов, которые требуется выбрать:

SELECT ВЫБИРАЕМЫЕ\_СТОЛБЦЫ FROM ИМЯ\_ТАБЛИЦЫ

Каждому столбцу можно задать псевдоним через оператор AS

Для выбора определённых строк таблицы вместе с оператором SELECT уже потребуется ключевое слово WHERE, указывающее на некоторое значение или несколько значений, содержащиеся в интересующих нас строках. Наиболее простые условия задаются при помощи операторов сравнения и равенства (<, >, =), а также ключевого слова IS. Условий может быть несколько, тогда они перечисляются с использованием ключевого слова AND. Запросы для выбора строк имеют следующий синтаксис:

SELECT ИМЯ\_СТОЛБЦА FROM ИМЯ\_ТАБЛИЦЫ WHERE УСЛОВИЕ

SELECT Dept, Name, Job

FROM Staff WHERE Job='Clerk' AND DEPT=38

SELECT ID, Name FROM Staff

WHERE Comm IS NULL

Для сортировки используется оператор ORDER BY, ASC для возрастания, DESC для убывания

SELECT ИМЯ\_СТОЛБЦА FROM ИМЯ\_ТАБЛИЦЫ WHERE УСЛОВИЕ

ORDER BY СТОЛБЕЦ, ПО КОТОРОМУ СОРТИРУЮТСЯ РЕЗУЛЬТАТЫ

SELECT Name, Job, Years

FROM Staff WHERE Dept=84

ORDER BY Years DESC

В SQL Select есть возможность добавлять подзапросы, то есть значения условия отбора возвращаются из другого запроса (вложенного запроса), начинающегося также с SELECT.

SELECT Division, Location

FROM Org

WHERE Deptnumb =

(SELECT Dept FROM Staff

WHERE ID = 16)

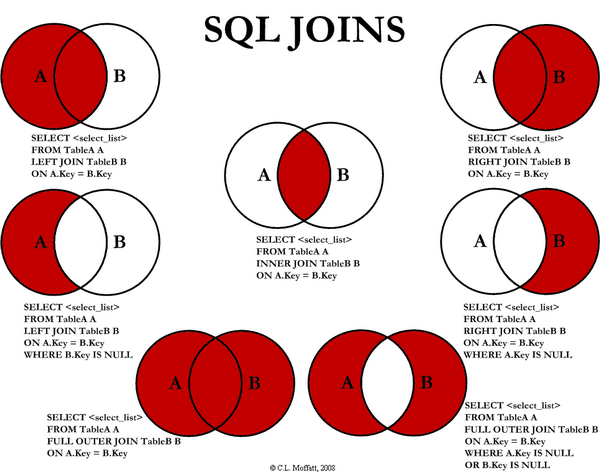
Когда для в запросе необходимы не все данные, а определенное количество используется оператор LIMIT

SELECT ID, Name FROM Staff

WHERE Comm IS NULL LIMIT 5

Указание двух цифр после оператора LIMIT позволяет пропустить N-ное количество строк, и взять необходимое количество

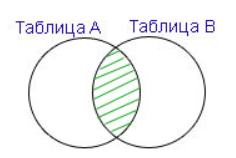
## 11. Оператор JOIN в SQL. Общий синтаксис. Виды объединений, их различия и области применений. Примеры.



**SQL JOIN** - соединение таблиц базы данных

**INNER JOIN или просто JOIN (внутреннее соединение)**

Запрос с оператором INNER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON.



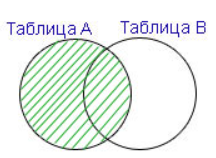
SELECT Parts.Part, Categories.Catnumb AS Cat, Categories.Price

FROM Parts INNER JOIN Categories

ON Parts.Cat = Categories.Catnumb

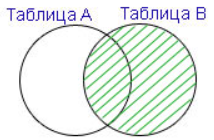
**LEFT OUTER JOIN (левое внешнее соединение)**

Запрос с оператором LEFT OUTER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из первой по порядку (левой) таблицы, даже если они не соответствуют условию. У записей левой таблицы, которые не соответствуют условию, значение столбца из правой таблицы будет NULL (неопределённым).



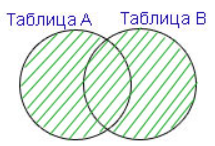
**RIGHT OUTER JOIN (правое внешнее соединение)**

Запрос с оператором RIGHT OUTER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из второй по порядку (правой) таблицы, даже если они не соответствуют условию. У записей правой таблицы, которые не соответствуют условию, значение столбца из левой таблицы будет NULL (неопределённым).



**FULL OUTER JOIN (полное внешнее соединение)**

Запрос с оператором FULL OUTER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из первой (левой) и второй (правой) таблиц, даже если они не соответствуют условию. У записей, которые не соответствуют условию, значение столбцов из другой таблицы будет NULL (неопределённым).



**Соединение нескольких таблиц:**

SELECT ИМЕНА\_СТОЛБЦОВ (1..N)

FROM ИМЯ\_ТАБЛИЦЫ\_1 JOIN ИМЯ\_ТАБЛИЦЫ\_2

ON УСЛОВИЕ

JOIN ИМЯ\_ТАБЛИЦЫ\_3

ON УСЛОВИЕ

...

JOIN ИМЯ\_ТАБЛИЦЫ\_M

ON УСЛОВИЕ

**CROSS JOIN (перекрестное соединение)**

Использование оператора SQL CROSS JOIN в наиболее простой форме - без условия соединения - реализует операцию декартова произведения в реляционной алгебре. Результатом такого соединения будет сцепление каждой строки первой таблицы с каждой строкой второй таблицы. Таблицы могут быть записаны в запросе либо через оператор CROSS JOIN, либо через запятую между ними.

SELECT (\*) Categories CROSS JOIN Parts



## 12. Использование групповых функций в SQL. Синтаксис оператора SELECT при использовании группировок. Примеры использования.

Синтаксис

SELECT *column\_name(s)*

FROM *table\_name*

WHERE *condition*

GROUP BY *column\_name(s)*

HAVING *condition*

Where - условие до агрегации, а в having можно использовать агрегирующие функции, чтобы отфильтровать полученные после агрегации данные.

Групповые функции могут использоваться в предложениях SELECT и HAVING

### Список

| AVG([DISTINCT | ALL] n) | Возвращает среднее значение n. |
| --- | --- |
| COUNT(\* | [DISTINCT|ALL] выражение) | Возвращает количество строк в запросе. Если указано *выражение*, подсчитываются только те строки, в которых значение *выражения* не пусто. |
| MAX([DISTINCT | ALL] выражение) | Возвращает максимальное значение *выражения*. |
| MIN([DISTINCT | ALL] выражение) | Возвращает минимальное значение *выражения*. |
| STDDEV([DISTINCT | ALL] n) | Возвращает стандартное отклонение n. |
| SUM([DISTINCT | ALL] n) | Возвращает сумму значений n. |
| VARIANCE([DISTINCT | ALL] n) | Возвращает дисперсию n. |

Пример использования

SELECT Фамилия, Avg(Результат) AS Средний\_балл FROM Результаты

GROUP BY [№ зач.книжки] HAVING Аvg(Результат) > 4.5

## 13. Реляционные базы данных. Нормализация отношений. Первая нормальная форма. Вторая нормальная форма. Третья нормальная форма. Другие нормальные формы.

Реляционными (от англ. relation — отношение) называются базы данных, в основе построения которых лежит реляционная модель.

Данные в реляционных структурах организованы в виде набора таблиц, называемых отношениями, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных.

Каждая строка таблицы представляет собой набор связанных значений, относящихся к одному объекту, или сущности. Каждая строка в таблице может быть помечена уникальным идентификатором, называемым первичным ключом, а строки из нескольких таблиц могут быть связаны с помощью внешних ключей.

Особенностью реляционной базы данных является использование в ней реляционной модели данных и вытекающие из этого последствия:

● Модель данных в реляционных БД определена заранее. Является строго типизированной, содержит ограничения и отношения для обеспечения целостности данных.

● Модель данных основана на естественном представлении содержащихся данных, а не на функциональности приложения.

● Модель данных подвергается нормализации, чтобы избежать дублирования данных. Нормализация порождает отношения между таблицами. Отношения связывают данные разных таблиц. Избыточность устраняется, как правило, за счёт декомпозиции отношений (таблиц), т.е. разбиения одной таблицы на несколько.

Нормальная форма — требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц).

**Первая нормальная форма**

Переменная отношения находится в первой нормальной форме тогда и только тогда, когда в любом допустимом значении этой переменной каждый кортеж отношения содержит только одно значение для каждого из атрибутов.

**Вторая нормальная форма**

Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа(ПК).

Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

**Третья нормальная форма**

Третья нормальная форма предполагает, что каждый столбец, не являющийся ключом, должен зависеть только от столбца, который является ключом, то есть должна отсутствовать транзитивная функциональная зависимость (Проще говоря, правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.)

При применении третьей нормальной формы таблица должна находиться во второй нормальной форме. 3NF позволяет значительно снизить избыточность данных.

**Другие нормальные формы**

Кроме рассмотренных в основной части данного занятия первых трех нормальных форм существует еще три нормальных формы: четвертая нормальная форма (4NF), нормальная форма Бойса-Кодда (BCNF) и пятая нормальная форма (5NF), которую иногда называют формой проекции-соединения (PJNF). Для понимания определений этих нормальных форм также необходимо знание понятий реляционной алгебры. Поэтому ограничимся перечислением основных свойств нормальных форм:

· каждая последующая нормальная форма в некотором смысле улучшает свойства предыдущей;

· при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

Следует также заметить, что переход к 5NF усложняет структуру БД, что затрудняет операции модификации, поэтому при проектировании БД имеет смысл оценить какие операции будут производиться чаще, чтобы определить, какая нормальная форма будет наиболее подходящей в данном конкретном случае. В простых информационных системах 5NF используется крайне редко – инженеры-проектировщики обычно ограничиваются 3NF.

#### Четвертая нормальная форма

Отношение находится в 4НФ, если оно находится в НФБК и все нетривиальные многозначные зависимости фактически являются функциональными зависимостями от ее потенциальных ключей.

В отношении R (A, B, C) существует многозначная зависимость R.A -> -> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от С.

Предположим, что рестораны производят разные виды пиццы, а службы доставки ресторанов работают только в определенных районах города. Составной первичный ключ соответствующей переменной отношения включает три атрибута: {Ресторан, Вид пиццы, Район доставки}.

Такая переменная отношения не соответствует 4НФ, так как существует следующая многозначная зависимость:

{Ресторан} → {Вид пиццы}

{Ресторан} → {Район доставки}

То есть, например, при добавлении нового вида пиццы придется внести по одному новому кортежу для каждого района доставки. Возможна логическая аномалия, при которой определенному виду пиццы будут соответствовать лишь некоторые районы доставки из обслуживаемых рестораном районов.

Для предотвращения аномалии нужно декомпозировать отношение, разместив независимые факты в разных отношениях. В данном примере следует выполнить декомпозицию на {Ресторан, Вид пиццы} и {Ресторан, Район доставки}.

Однако, если к исходной переменной отношения добавить атрибут, функционально зависящий от потенциального ключа, например цену с учётом стоимости доставки ({Ресторан, Вид пиццы, Район доставки} → Цена), то полученное отношение будет находиться в 4НФ и его уже нельзя подвергнуть декомпозиции без потерь.

#### Пятая нормальная форма

Отношения находятся в 5НФ, если оно находится в 4НФ и отсутствуют сложные зависимые соединения между атрибутами.

Если «Атрибут\_1» зависит от «Атрибута\_2», а «Атрибут\_2» в свою очередь зависит от «Атрибута\_3», а «Атрибут\_3» зависит от «Атрибута\_1», то все три атрибута обязательно входят в один кортеж.

Это очень жесткое требование, которое можно выполнить лишь при дополнительных условиях. На практике трудно найти пример реализации этого требования в чистом виде.

Например, некоторая таблица содержит три атрибута «Поставщик», «Товар» и «Покупатель». Покупатель\_1 приобретает несколько Товаров у Поставщика\_1. Покупатель\_1 приобрел новый Товар у Поставщика\_2. Тогда в силу изложенного выше требования Поставщик\_1 обязан поставлять Покупателю\_1 тот же самый новый Товар, а Поставщик\_2 должен поставлять Покупателю\_1, кроме нового Товара, всю номенклатуру Товаров Поставщика\_1. Этого на практике не бывает. Покупатель свободен в своем выборе товаров. Поэтому для устранения отмеченного затруднения все три атрибута разносят по разным отношениям (таблицам). После выделения трех новых отношений (Поставщик, Товар и Покупатель) необходимо помнить, что при извлечении информации (например, о покупателях и товарах) необходимо в запросе соединить все три отношения. Любая комбинация соединения двух отношений из трех неминуемо приведет к извлечению неверной (некорректной) информации. Некоторые СУБД снабжены специальными механизмами, устраняющими извлечение недостоверной информации. Тем не менее, следует придерживаться общей рекомендации: структуру базы данных строить таким образом, чтобы избежать применения 4НФ и 5НФ.

Пятая нормальная форма ориентирована на работу с зависимыми соединениями. Указанные зависимые соединения между тремя атрибутами встречаются очень редко. Зависимые соединения между четырьмя, пятью и более атрибутами указать практически невозможно.

## 14. Транзакции в SQL. Понятие транзакции, организация транзакций, вложенные транзакции. Транзакции и управление параллельной работой.[КВ3]

Транзакция — это набор операций по работе с базой данных (БД), объединенных в одну атомарную пачку. Или иными словами- это архив для запросов к базе. Он защищает ваши данные благодаря принципу «всё, или ничего».

**Как отправить транзакцию**

Чтобы обратиться к базе данных, сначала надо открыть соединение с ней. Это называется коннект (от англ. connection, соединение) Чтобы сгруппировать запросы в одну атомарную пачку, используем транзакцию. Транзакцию надо:

Открыть. Выполнить все операции внутри. Закрыть.

Далее можно как закрыть соединение с базой данных, так и не закрывать и продолжить работу с ней.

**Как открыть транзакцию**

Зависит от базы данных. В Oracle транзакция открывается сама, по факту первой изменяющей операции. А в MySql надо явно писать «start transaction».

**Как закрыть транзакцию**

Есть 2 варианта:

1. COMMIT — подтверждаем все внесенные изменения;
2. ROLLBACK — откатываем их;

И вся фишка транзакционной базы в том, что база сначала применяет запрос «виртуально», реально ничего в базе не изменив. Ты можешь посмотреть, как запрос изменит базу, ничего при этом не сохраняя. Если изменения не удовлетворяют- можно отменить их командой ROLLBACK, если все хорошо- принять командой COMMIT

Вложенные транзакции

Пример:

Есть готовые куски кода с транзакциями:

Транзакция 1: Пополнить счёт пользователя

Транзакция 2: Дать скидку пользователю

И есть транзакция, которая отвечает за регистрацию:

Транзакция 3: Регистрация (создать пользователя)

Пришла задача: дать пользователю денег и скидку при регистрации.

Можно добавить вызов первых двух транзакций в третью транзакцию и получится вложенность. То же самое можно выполнить и без вложенности, но незачем.

Стоит учесть, что вложенные транзакции не поддерживают MySQL и Oracle.

**Как обеспечить правильную работу транзакций, обращающихся к одним и тем же ресурсам?**

Два основных способа:

* Блокирование “общих” ресурсов;
* Предоставление транзакциям, конкурирующим за ресурсы, разных экземпляров данных.

**Блокирование**

Два типа блокировок:

* Монопольные блокировки (e**X**clusive locks или X-locks) не разрешают доступ другим транзакциям;
* Разделяемые блокировки (**S**hared locks или S-locks) разрешают совместный доступ.

Виды блокировок:

1. Блокировки различают **по размерам блокируемого ресурса** (поле записи, запись, отношение, страница (блок базы), группа отношений, вся база). В современных СУБД блокировка полей не применяется.
2. **По степени разделяемости** блокируемых ресурсов различают:
   1. Монопольные блокировки, называемые еще блокировками записи.
   2. Разделяемые блокировки, иначе блокировки чтения.

**Иерархичность блокировок**. Если объект верхнего уровня обладает блокировкой, то такой же блокировкой обладает его объект нижнего уровня.

## 15. Типы задач машинного обучения: классификация, регрессия, кластеризация. Обучение с учителем и без учителя. Машинное обучение как оптимизационная задача.

Машинное обучение (Machine Learning) это тренировка математической модели на исторических данных для того, чтобы прогнозировать какое-то событие или явление на новых данных. То есть попытка заставить алгоритмы программ совершать действия на основе предыдущего опыта, а не только на основе имеющихся данных.

* **регрессия** — предсказание числовых значений признаков, например, предсказание будущих объемов продаж на основании известных данных о продажах в прошлом;
* **классификация** — предсказание того, к какому из известных классов относится объект, например, предсказание того, вернет ли заемщик кредит, на основании данных о том, как возвращали кредиты заемщики в прошлом;

Процесс обучения модели — это подача данных для нейросети или модели МО, которая в результате должна вывести определённые шаблоны для данных. В процессе обучения модели с учителем на вход подаются признаки и метки, а при прогнозировании на вход классификатора подаются только признаки.

* **кластеризация** — разделение большого множества объектов на кластеры — классы, внутри которых объекты похожи между собой, например, сегментирование рынка, разделение всех потребителей на классы так, что внутри классов потребители похожи между собой, а в разных классах — отличаются;

### **С учителем**

В идеальном случае у нас есть исходные данные, т. е. правильные ответы для системы. Представьте, что нужно проанализировать тысячи квартир Санкт-Петербурга, зная все параметры: цену, район, этаж, площадь, наличие инфраструктуры. С помощью модели машинного обучения мы предскажем рыночную стоимость, т. е. решается регрессионная задача.

Или надо узнать, есть или нет рак у пациента, имея в наличии его медицинские показатели. Или понять, входящее эл. письмо — это спам или нет. Всё это задачи на классификацию.

### **Без учителя**

Здесь готовых ответов нет, но менее интересно не становится, даже наоборот. Представьте, что мы знаем рост и вес большой группы людей, в соответствии с чем нужно пошить одежду трёх видов. Это кластеризация (строгого и единственно верного деления тут нет).

Или у нас каждый из объектов выборки имеет сотни разных признаков. Отобразить графически такую выборку будет очень сложно, поэтому мы уменьшим число признаков, скажем, до 3-х. Это уменьшение размерности.

## 16. Регрессия как задача машинного обучения. Постановка задачи, математическая формализация. Целевая функция. Примеры.

**Регрессия** — предсказание числовых значений признаков, например, предсказание будущих объемов продаж на основании известных данных о продажах в прошлом.

**Мат. формализация**

Регрессия — зависимость математического ожидания (например, среднего значения) случайной величины от одной или нескольких других случайных величин (свободных переменных), то есть E(y|\mathbf{x})=f(\mathbf{x}).

Регрессионным анализом называется поиск такой функции f, которая описывает эту зависимость. Регрессия может быть представлена в виде суммы неслучайной и случайной составляющих:  y=f(\mathbf{x})+\nu, 

где f — функция регрессионной зависимости, а \nu — аддитивная случайная величина с нулевым матожиданием.

Критерием качества приближения (**целевой функцией**) обычно является среднеквадратичная ошибка: сумма квадратов разности значений модели и зависимой переменной для всех значений независимой переменной в качестве аргумента.

Примеры алгоритмов:

**Линейной регрессией** является линейная модель, например, модель, которая предполагает линейную связь между входными переменными (x) и единственной выходной переменной (y). Более конкретно, что у можно вычислить из линейной комбинации входных переменных (х).

Когда есть одна входная переменная (x), метод называется простой линейной регрессией, Когда есть несколько входных переменных , упоминается метод как множественная линейная регрессия.

**Логистическая регрессия** похожа на линейную, но выходное значение преобразуется с помощью нелинейной или логистической функции.

Логистическая функция выглядит как большая буква S и преобразовывает любое значение в число в пределах от 0 до 1. Это весьма полезно, так как мы можем применить правило к выходу логистической функции для привязки к 0 и 1 (например, если результат функции меньше 0.5, то на выходе получаем 1) и предсказания класса.

## 17. Классификация как задача машинного обучения. Постановка задачи, математическая формализация. Целевая функция. Примеры.

**Классификация** — предсказание того, к какому из известных классов относится объект, например, предсказание того, вернет ли заемщик кредит, на основании данных о том, как возвращали кредиты заемщики в прошлом;

Процесс обучения модели — это подача данных для нейросети, которая в результате должна вывести определённые шаблоны для данных. В процессе обучения модели с учителем на вход подаются признаки и метки, а при прогнозировании на вход классификатора подаются только признаки.

**Математическая постановка задачи** - Пусть X — множество описаний объектов, Y — конечное множество номеров (имён, меток) классов. Существует неизвестная целевая зависимость — отображение y^{*}:\; X\to Y, значения которой известны только на объектах конечной [обучающей выборки](http://www.machinelearning.ru/wiki/index.php?title=%D0%92%D1%8B%D0%B1%D0%BE%D1%80%D0%BA%D0%B0) X^m = \{(x_1,y_1),\dots,(x_m,y_m)\}. Требуется построить [алгоритм](http://www.machinelearning.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) a:\; X\to Y, способный классифицировать произвольный объект x \in X.

**Примеры алгоритмов:**

**Дерево решений** строит модели классификации или регрессии в форме древовидной структуры. Он использует набор правил if-then, который является взаимоисключающим и исчерпывающим для классификации. Правила выучены последовательно, используя данные обучения по одному. Каждый раз, когда правило выучено, кортежи, на которые распространяются правила, удаляются. Этот процесс продолжается на тренировочном наборе до достижения условия завершения.

Дерево построено рекурсивным способом сверху вниз. Все атрибуты должны быть категоричными. В противном случае они должны быть заранее дискретизированы. Атрибуты в верхней части дерева оказывают большее влияние на классификацию и идентифицируются с использованием концепции получения информации.

**Наивный байесовский метод** - вероятностный классификатор, основанный на теореме Байеса в простом предположении, согласно которому атрибуты условно независимы. Чаще всего он используется либо как примитивный эталон для сравнения различных моделей алгоритмов

***К*Ближайших соседей** - Алгоритм KNN можно разделить на две простые фазы: [обучения](https://wiki.loginom.ru/articles/training.html) и классификации. При обучении алгоритм просто запоминает векторы признаков наблюдений и их [метки классов](https://wiki.loginom.ru/articles/class-label.html) (т.е. примеры). Также задаётся параметр алгоритма *k*, который задаёт число «соседей», которые будут использоваться при классификации.

На фазе классификации предъявляется новый объект, для которого метка класса не задана. Для него определяются k ближайших (в смысле некоторой метрики) предварительно классифицированных наблюдений. Затем выбирается класс, которому принадлежит большинство из *k* ближайших примеров-соседей, и к этому же классу относится классифицируемый объект.

***Примеры задач:***

*Задачи медицинской диагностики*

В роли объектов выступают пациенты. Признаки характеризуют результаты обследований, симптомы заболевания и применявшиеся методы лечения. Примеры бинарных признаков: пол, наличие головной боли, слабости. Порядковый признак — тяжесть состояния (удовлетворительное, средней тяжести, тяжёлое, крайне тяжёлое).

*Предсказание месторождений полезных ископаемых*

Признаками являются данные геологической разведки. Наличие или отсутствие тех или иных пород на территории района кодируется бинарными признаками. Физико-химические свойства этих пород могут описываться как количественными, так и качественными признаками. Обучающая выборка составляется из прецедентов двух классов: районов известных месторождений и похожих районов, в которых интересующее ископаемое обнаружено не было.

Оптическое распознавание символов

Распознавание речи

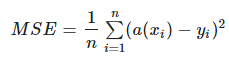
Обнаружение спама

## 18. Метрики качества результатов машинного обучения. Метрики качества моделей регрессии и классификации. MAE, MSE, Precision, Recall, F1, ROC-AUC.

**Регрессия**:

**Средняя квадратичная ошибка (англ. Mean Squared Error, MSE)**

MSE применяется в ситуациях, когда нам надо подчеркнуть большие ошибки и выбрать модель, которая дает меньше больших ошибок прогноза. Грубые ошибки становятся заметнее за счет того, что ошибку прогноза мы возводим в квадрат. И модель, которая дает нам меньшее значение среднеквадратической ошибки, можно сказать, что что у этой модели меньше грубых ошибок.



**Cредняя абсолютная ошибка (англ. Mean Absolute Error, MAE)**



MSE сильнее штрафует за большие отклонения по сравнению с MAE, и поэтому более чувствителен к выбросам. При использовании любого из этих двух может быть полезно проанализировать, какие объекты вносят наибольший вклад в общую ошибку — не исключено, что на этих объектах была допущена ошибка при вычислении признаков или целевой величины.

Среднеквадратичная ошибка подходит для сравнения двух моделей или для контроля качества во время обучения, но не позволяет сделать выводов о том, на сколько хорошо данная модель решает задачу. Например, MSE = 10 является очень плохим показателем, если целевая переменная принимает значения от 0 до 1, и очень хорошим, если целевая переменная лежит в интервале (10000, 100000).

**Классификация**:

**Точностью** (**precision**) называется доля правильных ответов модели в пределах класса — это доля объектов действительно принадлежащих данному классу относительно всех объектов которые система отнесла к этому классу.



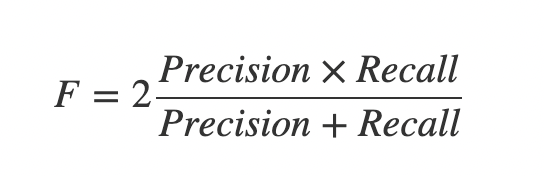
Именно введение precision не позволяет нам записывать все объекты в один класс, так как в этом случае мы получаем рост уровня False Positive.

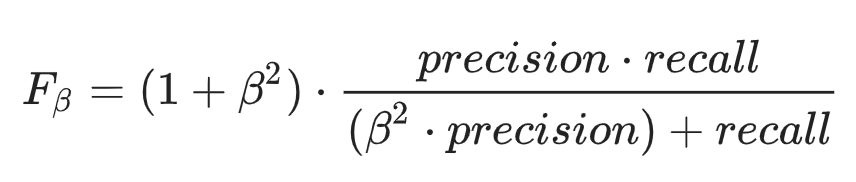
**Полнота (recall)** — это доля истинно положительных классификаций. Полнота показывает, какую долю объектов, реально относящихся к положительному классу, мы предсказали верно.



Полнота (recall) демонстрирует способность алгоритма обнаруживать данный класс вообще.

**F-мера** (в общем случае Fβ) — среднее гармоническое **precision** и **recall**:

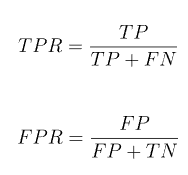




F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю.

В sklearn есть функция \_metrics.classification*report*, возвращающая **recall**, **precision** и **F-меру** для каждого из классов, а также количество экземпляров каждого класса.

Одним из способов оценить модель в целом, не привязываясь к конкретному порогу, является **AUC-ROC** (или ROC AUC) — площадь (Area Under Curve) под кривой ошибок (Receiver Operating Characteristic curve ). Данная кривая представляет из себя линию от (0,0) до (1,1) в координатах True Positive Rate (TPR) и False Positive Rate (FPR):



TPR нам уже известна, это полнота, а FPR показывает, какую долю из объектов negative класса алгоритм предсказал неверно. В идеальном случае, когда классификатор не делает ошибок (FPR = 0, TPR = 1) мы получим площадь под кривой, равную единице; в противном случае, когда классификатор случайно выдает вероятности классов, AUC-ROC будет стремиться к 0.5, так как классификатор будет выдавать одинаковое количество TP и FP.

Каждая точка на графике соответствует выбору некоторого порога. Площадь под кривой в данном случае показывает качество алгоритма (больше — лучше), кроме этого, важной является крутизна самой кривой — мы хотим максимизировать TPR, минимизируя FPR, а значит, наша кривая в идеале должна стремиться к точке (0,1).

## 19. Кластеризация как задача машинного обучения. Метод ближайших соседей (kNN) и его обобщения. Подбор числа k. Методы метрической классификации в библиотеке Scikit-Learn.

**Кластеризация** (англ. *cluster analysis*) — задача группировки множества объектов на подмножества (**кластеры**) таким образом, чтобы объекты из одного кластера были более похожи друг на друга, чем на объекты из других кластеров по какому-либо критерию.

Задача кластеризации относится к классу задач обучения без учителя.

Некоторые сферы, в которых решение задачи кластеризации помогает решать многие прикладные задачи.

* Разделение клиентов и покупателей на группы по среднему размеру стоимости покупки или другим признакам.
* Определение сегментов рынка по какому-либо показателю потребности в товаре или услуге.
* Определение групп посетителей социальных сетей по показателям активности.
* Выделение групп товаров по показателям оборачиваемости или другим показателям.
* Определение групп объектов рынка информации - сайтов, каналов, сюжетов по какому-либо признаку, например, длительности просмотра.
* Определение групп пациентов по каким-либо признакам течения заболевания.

### Метод k-ближайших соседей

Относит объекты к классу, которому принадлежит большинство из K его ближайших соседей в многомерном пространстве признаков.

Число k — это количество соседних объектов в пространстве признаков, которые сравниваются с классифицируемым объектом. При k = 10, каждый объект сравнивается с 10-ю соседями.

В процессе обучения алгоритм просто запоминает все векторы признаков и соответствующие им метки классов. При работе с реальными данными, т. е. наблюдениями, метки класса которых неизвестны, вычисляется расстояние между вектором нового наблюдения и ранее запомненными. Затем выбирается k ближайших к нему векторов, и новый объект относится к классу, которому принадлежит большинство из них.

Выбор **параметра k противоречив**. Увеличение его значения повышает достоверность классификации, но при этом границы между классами становятся менее четкими. На практике **хорошие результаты дают эвристические методы выбора параметра k**, например, перекрестная проверка (кросс-валидация).

Кросс-валидация позволяет избежать перекоса данных в процессе обучения (когда данных одного класса значительно больше данных другого)

Разновидности:

* **Метод ближайшего соседа**. Классифицируемый объект относится к тому классу , которому принадлежит ближайший объект обучающей выборки .
* **Метод ближайших соседей**. Для повышения надежности классификации объект относится к тому классу, которому принадлежит большинство из его соседей — ближайших к нему объектов обучающей выборки . В задачах с двумя классами число соседей берут нечётным, чтобы не возникало ситуаций неоднозначности, когда одинаковое число соседей принадлежат разным классам.
* **Метод взвешенных ближайших соседей.** В задачах с числом классов 3 и более нечётность уже не помогает, и ситуации неоднозначности всё равно могут возникать. Тогда n-му соседу приписывается вес , как правило, убывающий с ростом ранга соседа . Объект относится к тому классу, который набирает больший суммарный вес среди ближайших соседей.

В библиотеке scikit learn классификатор knn выглядит так:

>>> X = [[0], [1], [2], [3]]

>>> y = [0, 0, 1, 1]

>>> from sklearn.neighbors import KNeighborsClassifier

>>> neigh = KNeighborsClassifier(n\_neighbors=3)

>>> neigh.fit(X, y)

KNeighborsClassifier(...)

>>> print(neigh.predict([[1.1]]))

[0]

>>> print(neigh.predict\_proba([[0.9]]))

[[0.66666667 0.33333333]]

Остальные классификаторы из sklearn:

from sklearn.linear\_model import LogisticRegression # Логистическая регрессия

from sklearn.naive\_bayes import GaussianNB # Баес

from sklearn.tree import DecisionTreeClassifier # Дерево решений

from sklearn.svm import SVC # Опорные вектора

## 20. Структура проекта нативного Android приложения. Основные файлы. Манифест приложения. Файлы с разметкой. Файлы ресурсов. Использование альтернативных ресурсов. Используемые языки для разработки и описания приложения.

Для начала существует папка проекта – название совпадает с названием самого проекта.

Если в папке проекта – все папки, которые содержатся в нем.

Папка app включает подпапки:

* src
  + manifest
    - ***AndroidManifest -*** информация о пакетах приложения, компонентах типа Activity, Service
  + main
    - java - исходный код приложения
    - res - файлы ресурсов
      * ***drawable.*** файлы с изображениями
      * ***layout.*** располагает xml файлами, которые используются для построения пользовательского интерфейса Android приложения
      * ***menu*** находятся xml файлы, используемые только для создания меню
      * ***mipmap*** папке хранятся только значки приложения
      * ***values*** *константы*
      * ***font*** *шрифты*
* build
* libs

**src + manifest**

Файл ***AndroidManifest.xml*** является одним из самых важных в Android проекте. В нем содержится информация о пакетах приложения, компонентах типа Activity, Service и т.д. Он задает конфигурацию приложения: объявляет компоненты приложения, перечисляет любые библиотеки, связанные с приложением (помимо библиотек Android, связанных по умолчанию), и объявляет разрешения, которые требуются для работы приложения (например, доступ в сеть, разрешение на отправку SMS и т. д.).

Файл ***AndroidManifest.xml*** выполняет следующие задачи:

* Предоставляет разрешения приложению на использование или доступ к другим компонентам системы. Например : получить доступ на чтение SMS в телефоне или получить доступ к геолокации
* Определяет, как будут запускаться, например, Activity (какие фильтры использовать).

Простыми словами – это список основной информации о самом приложении, чтобы наш телефон/планшет знали, что им делать с этим приложением. Содержит в себе огромное количество настроек.

**src + main**

* Папка ***java*** содержит исходный код приложения. Классы могут быть расположены в различных пакетах, но обязательно внутри папки java.
* В папке ***res*** (ресурсы) расположены все используемые приложением ресурсы, включая изображения, различные xml файлы, анимации, звуковые файлы и многие другие. Внутри папки res эти все ресурсы распределены по своим папкам:
  + Папка ***drawable*** содержит файлы с изображениями, которые будет использоваться в приложении. Для изображений (PNG, JPEG и т. д.). Изображения в каждой папке рассчитаны на соответствующее разрешение экрана мобильного устройства;
  + Папка ***layout*** располагает xml файлами, которые используются для построения пользовательского интерфейса Android приложения. (хорошим стилем программирования является создание отдельных ресурсов и использование ссылок на них)
  + В папке ***menu*** находятся xml файлы, используемые только для создания меню.
  + В ***mipmap*** папке хранятся только значки приложения.
  + ***values*** хранит те xml файлы, в которых определяются простые значения типа строк, массивов, целых чисел, размерностей, цветов и стилей.

**build**

Папка служит для обработки устройством приложения. Важный файл – ***r***, есть в каждом приложении.

**libs**

***libs*** – каталог с библиотеками. Если Вам понадобятся какие – то дополнительные библиотеки, их необходимо будет добавить в эту директорию и обновить проект.

* Скрипты Gradle используются для автоматизации сборки проекта. Android Studio выполняет сборку приложения в фоновом режиме без какого-либо вмешательства со стороны разработчика. Этот процесс сборки осуществляется с использованием системы Gradle — инструментария для автоматической сборки с помощью набора конфигурационных файлов. Gradle скрипты написаны на языке groove.
* Среда разработки Eclipse использует файл project.properties для настройки метаданных проекта. В Android Studio этим занимается папка .idea. Это означает, что метаданные конкретного проекта хранятся в Android Studio.

**Ресурсы приложений**

Ресурс в приложении Android представляет собой файл, например, файл разметки интерфейса или некоторое значение, например, простую строку. То есть ресурсы представляют собой и файлы разметки, и отдельные строки, и звуковые файлы, файлы изображений и т.д. Все ресурсы находятся в проекте в каталоге res. Для различных типов ресурсов, определенных в проекте, в каталоге res создаются подкаталоги.

Альтернативные ресурсы могут быть добавлены средствами IDE или руками - загружаются в соответствующую папку ресурса в виде xml, png, jpg, ttf и тд файлов.

**Виды ресурсов**

Поддерживаемые подкаталоги:

* **animator/:** xml-файлы, определяющие анимацию свойств
* **anim/:** xml-файлы, определяющие tween-анимацию
* **color/:** xml-файлы, определяющие список цветов
* **drawable/:** Графические файлы (.png, .jpg, .gif)
* **mipmap/:** Графические файлы, используемые для иконок приложения под различные разрешения экранов
* **layout/:** xml-файлы, определяющие пользовательский интерфейс приложения
* **menu/:** xml-файлы, определяющие меню приложения
* **raw/:** различные файлы, которые сохраняются в исходном виде
* **values/:** xml-файлы, которые содержат различные используемые в приложении значения, например, ресурсы строк
* **xml/:** Произвольные xml-файлы
* **font/:** файлы с определениями шрифтом и расширениями .ttf, .otf или .ttc, либо файлы XML, который сод

### **Доступ к ресурсам**

**Ссылка на ресурс в коде**

Тип ресурса в данной записи ссылается на одно из пространств (внутренних классов), определенных в файле R.java, которые имеют соответствующие им типы в xml (например *string*, *layout*, *drawable* и т.д.)

**Ссылка на ресурс в xml**

Нередко возникает необходимость ссылаться на ресурс в файле xml, например, в файле, который определяет визуальный интерфейс, к примеру, в *activity\_main.xml*. Ссылки на ресурсы в файлах xml имеют следующую формализованную форму: *@[имя\_пакета:]тип\_ресурса/имя\_ресурса*

* *имя\_пакета* представляет имя пакета, в котором ресурс находится (указывать необязательно, если ресурс находится в том же пакете)
* *тип\_ресурса* представляет подкласс, определенный в классе *R* для типа ресурса
* *имя\_ресурса* имя файла ресурса без расширения или значение атрибута *android:name* в XML-элементе (для простых значений).

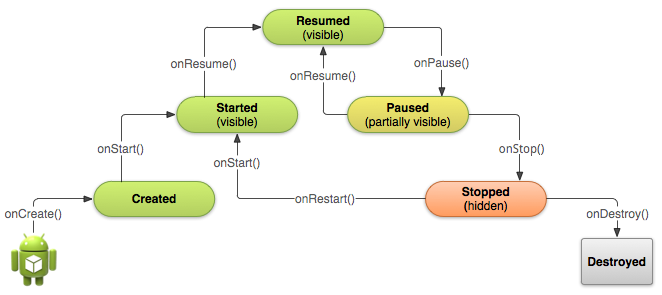
## **Основные языки программирования**

* Java, разработанная в 1990-х годах и по-прежнему наиболее востребованная, является золотым стандартом в области веб-разработки во всем мире, во всех областях. Она ориентирована на объекты и работает на любой платформе, что делает ее чрезвычайно универсальной. Если вы хотите, чтобы ваш safe можно было использовать практически во всех технологических компаниях в мире, то непременно выбирайте Java.
* Kotlin - фактически синтаксический сахар для джавы. Также реализован интересный подход с корутинами (coroutines) - особая функциональность, предоставляющая простой и легковесный способ написания параллельных приложений (в общем, реальная и красивая параллельность).

## 21. Компоненты Android приложений. Основные виды компонентов, их особенности и назначение. Жизненный цикл компонентов и использование методов жизненного цикла.

1. Activity - интерфейсы
2. Services - компоненты в фоновом режиме (музыка)
3. Content providers- данные, доступные другим приложениям (если есть доступ)
4. Broadcast receivers - отслеживание системных событий, их обработка
5. Activity – основной компонент в андроид- приложении. Активити – это экран, то что видит и с чем взаимодействует пользователь. Может содержать несколько View и фрагментов. Так же как и другие базовые компоненты, может быть точкой входа в приложение. Активити с MAIN Intent Filter в манифесте запускается при старте приложения, когда пользователь кликает на иконку приложения.
6. Все activity текущего приложения работают вместе и формируют единый пользовательский интерфейс, однако при этом они независимы между собой. Каждое из них реализовано как подкласс базового класса Activity, обеспечивающего создание окна, в котором программист может поместить визуальный интерфейс
7. Services(сервисы) представляют из себя компоненты, которые работают в фоновом режиме. Он, как правило, требуется для длительных операций или для обеспечения работы удаленных процессов, но в общем случае это просто режим, который функционирует, когда приложение не в фокусе. Примером такого процесса может стать прослушивание музыки в то время, когда пользователь делает что-то другое или получение данных по сети без блокирования текущей активности. Сервис сам по себе не предоставляет пользовательского интерфейса, то есть с пользователем не взаимодействует, а запускается, управляется и связан с другими компонентами, например, activity. Также может запускаться вместе с системой.
8. Content providers Данный компонент управляет наборами данных, которые приложения предоставляют другим. Эти данные могут храниться в файловой системе, базах данных SQLite, в сети, или в любом другом постоянном месте, к которому приложение может иметь доступ. Посредством content provider другое приложение может запрашивать данные и, если выставлены соответствующие разрешения, изменять их. Например, система Android содержит content provider, который управляет пользовательской информацией о контактах. Он позволяет любому приложению, обладающему соответствующими правами вызывать составляющие этого компонента для того, чтобы считывать, записывать или изменять информацию о конкретном человеке.
9. Broadcast receivers. Этот компонент отвечает за распространение общесистемных сообщений, отслеживание и реагирование на действия. Многие оповещения идут от системы, например, сообщения о том что заряд батареи мал или экран выключен. Приложения также могут инициировать такие оповещения, например, сигнализировать о том, что информация загружена на устройство и доступна к использованию. Как и сервисы, broadcast receiver не предоставляет пользовательского интерфейса, однако, он способен создавать уведомления в строке состояния, чтобы предупреждать пользователя о том, что произошло какое-то событие.

**Жизненный цикл**



**Методы на пикче**

## 22. Хранение данных в Android приложении. Виды хранилищ данных, их особенности и использование из приложения.

Виды:

Общие настройки  
Внутреннее и Внешнее хранилище  
Базы данных SQLite

**Общие настройки (SharedPreferences)**

Это постоянное хранилище на платформе Android, используемое приложениями для хранения своих настроек в виде пар ключ-значение, даже если пользователь закроет приложение, то эти настройки сохранятся, данные хранятся в XML-файле. Это хранилище является относительно постоянным, пользователь может зайти в настройки приложения и очистить данные приложения, тем самым очистив все данные в хранилище. Это самое простое хранилище, предполагающее хранение только базовой информации, а не больших объемов пользовательских данных. Например, если мы делаем конвертер валют, то можно записать в это хранилище основную валюту пользователя, чтоб каждый раз он ее не выбирал.

getPreferences() - для доступа к файлу настроек приложения

**Внутреннее и внешнее хранилище**

*Внутреннее хранилище*

это хранилище, которое недоступно пользователю, кроме как через установленные приложения, либо через root-права. То есть хоть оно и базируется на внутренней памяти устройства, данные оттуда не доступны для просмотра из файлового проводника (если, конечно, нет root, но это все же нестандартное поведение) и тд.

*Имеет следующие особенности:*

● Сохраненные здесь файлы доступны только вашему приложению.

● Когда пользователь удаляет ваше приложение, система удаляет все

файлы вашего приложения из внутреннего хранилища.

openFileOutput (FILENAME, MODE) - открывает файл для записи, где FILENAME - имя файла, а MODE - режим доступа к файлу (доступен только нашему приложения или всем и тд)

openFileInput (FILENAME) - открывает файл для чтения

*Внешнее хранилище (бывает двух типов):*

*Первичное внешнее хранилище***:** встроенное общее хранилище, которое доступно пользователю через файловый проводник, то есть свободный доступ к файлам без root-прав. Пример: внутренняя память устройства.

*Вторичные внешние хранилище:* съемный накопитель. Пример: SD-карта.

Имеют следующие особенности:

● Он доступен для чтения всем, поэтому сохраненные здесь файлы могут

быть прочитаны вне вашего контроля.

● Когда пользователь удаляет ваше приложение, система удаляет файлы

вашего приложения отсюда, только если вы сохраните их в каталоге из

getExternalFilesDir ().

getExternalFilesDir () - вернет путь к файловой директории во внешнем хранилище ассоциированную с вашим приложением (ассоциация создается самой системой), благодаря этому в случае удаления приложения эта папка также удалится, так как она имеет ассоциацию.

getExternalStorageDirectory ()- вернет просто путь к корню внешнего хранилища, без ассоциаций. Так что, если мы удалим приложение, то данные записанные таким образом не будут удалены.

**SQLite**

Sqlite используется для хранения более структурированных данных локально в Android устройстве.

В приложении, при подключении к БД мы указываем **имя** БД и **версию**. При этом могут возникнуть следующие ситуации:

1) БД **не существует**. Это может быть например в случае первичной установки программы. В этом случае приложение должно само **создать** БД и все таблицы в ней. И далее оно уже работает с только что созданной БД.

2) БД **существует**, но ее версия **устарела**. Это может быть в случае обновления программы. Например новой версии программы нужны дополнительные поля в старых таблицах или новые таблицы. В этом случае приложение должно **апдейтить** существующие таблицы и создать новые, если это необходимо.

3) БД **существует** и ее версия **актуальна**. В этом случае приложение успешно **подключается** к БД и работает.

Основные вспомогательные классы для взаимодействия с SQLite:

SQLiteOpenHelper - управление созданием, подключением и обновлением БД

SQLiteDatabase - методы для взаимодействия с БД (CRUD)

## 23. Виды мобильных приложений - нативные, веб, гибридные. Сравнительная характеристика. Используемые технологии и языки программирования.[КВ5]

**Нативные**.

Преимущества:

* Высокая производительность (скорость работы и надежность). Приложение создается для каждого ОС с учетом его особенностей.
* Имеют доступ к устройству пользователя и таким функциям как списки контактов, камера, NFC и др.
* Более безопасны с точки зрения защиты данных пользователя.

Недостатки:

* Стоимость разработки. Нативное приложение нужно разрабатывать отдельно для каждой ОС, а потом поддерживать его.
* Нативные приложения занимают место в памяти устройства пользователя, причем с каждым новым обновлением это занятое место может расти.

Веб. Адаптированные веб-сайты, которые открываются через браузеры. Создаются с помощью HTML/CSS/JavaScript

Преимущества:

* Не требуют настроек под операционную систему
* Не требуют обновлений, поэтому в теории их проще поддерживать со стороны разработчика

Недостатки:

* Веб-приложения зависят от браузера. И функции, которые доступны в одном браузере, могут не поддерживаться в другом.
* Сложности с добавлением на рабочий стол для быстрого доступа

Гибридные приложения. Веб-приложения, которые выглядят как нативные и имеют их признаки

Преимущества:

* Более бюджетная и быстрая разработка по сравнению с нативными.
* Возможность взаимодействия с ОС устройства.

Недостатки: Не такая хорошая производительность, как у нативных приложений.

## 24. Библиотеки Python для обработки больших данных. Обработка данных, превышающих объем доступной памяти. Библиотека Dask.

Обработка данных, превышающих объем доступной памяти:

1. Сокращение используемой памяти путем оптимизации типов данных

При загрузке данных с использованием Pandas типы определяются автоматически (если не указана иная задача). В большинстве случаев этот подход работает отлично, но выводимый тип не обязательно будет оптимизирован. Более того, если числовой столбец содержит отсутствующие значения, то автоматически вычисляемый тип будет float.

1. Разделение данных на фрагменты

Когда данные слишком велики, чтобы поместиться в памяти, можно воспользоваться опцией Pandas chunksize. Она позволяет разделить данные на фрагменты вместо того, чтобы работать с одним большим блоком. При использовании этой опции создается объект-итератор, с помощью которого можно просматривать различные фрагменты и выполнять фильтрацию или анализ точно так же, как при загрузке полного набора данных.

1. Использование преимуществ ленивых вычислений

Ленивые вычисления относятся к стратегиям, позволяющим отложить операцию вычисления до того момента, когда возникнет реальная необходимость в результате. Ленивые вычисления  —  важная концепция, особенно полезная для функционального программирования.

На основе ленивых вычислений построены такие механизмы распределенных вычислений, как Spark и Dask. Они предназначены для работы с кластерами, но пригодятся и для обработки больших массивов данных на компьютере.

В отличие от Pandas, они не загружают данные непосредственно в память. Вместо этого, во время чтения происходит сканирование данных, вывод их типов и разбиение их на разделы. Вычислительные графы для этих разделов строятся независимо и выполняются только тогда, когда это действительно необходимо (отсюда и название “ленивые вычисления”).

Dask – это гибкая библиотека для параллельных вычислений на Python. Она прекрасно взаимодействует с другими открытыми проектами, такими как NumPy, Pandas, и scikit-learn. В Dask есть структура массивов, которая эквивалентна массивам в NumPy, датафреймы в Dask аналогичны датафреймам в Pandas, а Dask-ML – это аналог scikit-learn.

Преимущество использования Dask заключается в том, что можно масштабировать вычисления до нескольких ядер на вашем компьютере. Получается возможность работать с большими объемами данных, которые не помещаются в память. Также можно ускорить вычисления, которые обычно занимают много места.

## 25. Библиотеки Python для обработки данных. Numpy, pandas - основные понятия, возможности, назначение, примеры использования. Основные форматы структурированных данных.

**NumPy** — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами. Библиотека написана на языках C и Fortran.

* Научные вычисления.
* Создание новых массивных библиотек.
* Data Science.
* Machine Learning.
* Визуализация данных.

При помощи Numpy можно совершать такие операции:

* Создание векторов и матриц
  + np.arange()
    - >>> np.arange(1, 5, 0.5)

array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])

* + np.matrix()
    - >>> b = np.array([[5, 6], [7, 8]])

>>> np.matrix(b)

matrix([[5, 6],

[7, 8]])

* + np.zeros(), np.eye()
    - Zeros создает нулевую матрицу, Eye - единичную квадратную
* Работа с матрицами и векторами
  + Сложение матриц
    - A + B
  + Вычитание матриц
    - A - B
  + Умножение матрицы на число
    - 3 \* A
  + Умножение матриц
    - A.dot(B)
  + Транспонирование матриц
    - A.T
* И др.
  + np.ravel()
    - преобразовать матрицу в одномерный вектор.
  + np.where()
    - возвращает один из двух заданных элементов в зависимости от условия.



* + np.random.permutation()
    - либо генерирует список заданной длины из натуральных чисел от нуля до указанного числа, либо перемешивает переданный ей в качестве аргумента массив.

**Pandas**:

Программная библиотека на языке Python для обработки и анализа данных. Работа pandas с данными строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. Предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временны́ми рядами. Использует для работы форматы Series и DataFrame. Если Series представляет собой одномерную структуру, которую для себя можно представить как таблицу с одной строкой, то DataFrame – это уже двумерная структура – полноценная таблица с множеством строк и столбцов.

**Работа с пандасом:**

* Импорт данных
  + Загрузка данных
    - df = pd.read\_csv('database/a.csv')
  + Создание датафрейма из данных, введённых вручную
    - df = pd.DataFrame([[1,'Bob', 'Builder'],

[2,'Sally', 'Baker'],

[3,'Scott', 'Candle Stick Maker']],

columns=['id','name', 'occupation'])

* + Копирование датафрейма
    - df2 =df.copy(deep=True)
* Экспорт данных
  + Экспорт
    - df.to\_csv('saved\_ratings.csv', index=False)
* Просмотр и исследование данных
  + df.head(3) - первые 3 записи
  + len(df) - кол-во строк
  + df.info() - сведения о датафрейме
  + df.describe() - вывод статистических данных
  + df.type.value\_counts() - подсчет кол-ва значений
* Извлечение информации из датафреймов
  + df.columns.tolist() - получение списка значений столбцов
* Добавление данных в датафрейм и удаление их из него
  + df.append(df[‘summa’], ignore\_index=True)
* Комбинирование датафреймов
  + concat, merge
    - rating.merge(df, left\_on=’df1\_id’, right\_on=’df2\_id’, suffixes=(‘\_left’, ‘\_right’))
* Фильтрация
  + df.iloc[0:3]
  + df[df['rating'] > 8]
* Сортировка
  + df.sort\_values('rating', ascending=False)
* Агрегирование
  + df.groupby('type').count()
* Очистка данных
* И др.