# 1. Рынок мобильных устройств. Современное состояние, история, перспективы.

Ответ:

Рынок мобильных устройств – это рынок, на котором объектом купли-продажи выступают мобильные устройства и аксессуары к ним.

Сегодня на рынке мобильных устройств представлен большой выбор моделей.

Существует множество мобильных устройств:

* Смартфоны
* Планшеты
* Носимые устройства
* Бортовые компьютеры

История:

В 1993 году популярными устройствами были Motorola, Nokia и NEC. Через 7 лет на рынке появились такие устройства, как Ericsson, Samsung, Panasonic, Alcatel. В 2005 самыми популярными устройствами были Nokia, Motorola и Samsung. Мобильные устройства, которые набирали популярность были LG, Sony Ericsson. Через 5 лет начали набирать популярность устройства Apple. Через несколько лет начали набирать популярность Microsoft, Lenovo, Huawei/Honor и Xiaomi. Сейчас самые популярные бренды - Samsung, Apple, Huawei/Honor и Xiaomi.

Современные торговые предприятия борются за каждого потребителя, пытаясь удовлетворить их потребности. Именно поэтому мобильные устройства были разбиты по ценовым категориям: бюджетные, средней ценовой категории, флагманы. К бюджетным устройствам можно отнести марки: Xiaomi, Honor, Realme, Oppo. К средней ценовой категории можно отнести Samsung, Huawei, Xiaomi. А к флагманам Samsung, Apple, Oneplus. Чаще всего в настоящее время люди покупают телефоны средней ценовой категории из-за своей стоимости и качества сборки.

Перспективы:

Мобильные устройства будут менятся не только в дизайне, но и в железе, так как прогресс не стоит на месте. Будут создаваться новые приложения и функции, а спрос падать не будет. Именно поэтому рынок мобильных устройств будет увеличиваться и развиваться.

# 2. Мобильные операционные системы. История, семейства, особенности.

Ответ:

Мобильные операционные системы - ОС для смартфонов, планшетов, карманных компьютеров и прочих портативных устройств.

Мобильные ОС сочетают в себе функциональность операционных систем для ПК с функциями для портативных устройств - сенсорный экран, сотовая связь, камера, GPS-навигация Wi-Fi, Bluetooth и т. д. Одна из важных задач, которую они решают - работа с экономией заряда батареи.

Портативные устройства мобильной связи (например, смартфоны) содержат две операционные системы. Основную программную платформу взаимодействия с пользователем дополняет вторая, низкоуровневая проприетарная операционная система реального времени, обслуживающая радиооборудование.

Историю мобильных ОС можно разбить на 4 этапа:

1980 - 1991: ОС для Программируемых калькуляторов и первых карманных компьютеров

1991 - 1998: ОС для PDA (personal digital assistance), карманных компьютеров, управляемых посредством стилуса

1998 - 2007: ОС для устройств, сочетающих функции КПК и коммуникатора (телефона). Представители - Palm OS, Windows CE, Symbian OS (Nokia OS)

2007 - наст. время: ОС для смартфонов и умных гаджетов. В текущем поколении производительность смартфонов сравнима с производительностью стационарных. Главные семейства мобильных ОС сегодня Android, iOS (iPadOS), Harmony OS (Huawei)

Отдельно про Android: 2003 - основание компании Android Inc. 2005 - поглощение Android Inc. поисковым гигантом Google. 23 сентября 2008 выход первой версии ОС Android – Android 1.0 Apple Pie

# 3. Особенности разработки приложений для мобильных устройств. Специфика работы мобильных операционных систем.

Ответ: Прям из презы Коротеева:

Особая схема дистрибуции приложений

Поддержка разнообразного оборудования

Высокая вариативность условий использования

Разработка под нецелевой операционной системой

Малый размер экрана

Особенности пользовательского интерфейса

Принцип разделения кода и ресурсов

Представление приложения как связанного набора компонентов

Автоматизация сборки приложения

Открытый исходный текст платформы

Мобильные операционные системы сочетают в себе функциональность Операционных Систем для ПК с функциями для мобильных и карманных устройств: сенсорный экран, сотовая связь, Bluetooth, Wi-Fi, GPS-навигация, камера, видеокамера, распознавание речи, диктофон, музыкальный плеер, NFC и инфракрасное дистанционное управление.

Портативные устройства мобильной связи (например, смартфоны) содержат две операционные системы. Основную программную платформу взаимодействия с пользователем дополняет вторая, низкоуровневая проприетарная операционная система реального времени, обслуживающая радиооборудование.

# 4. Особенности дистрибуции мобильных приложений.

Ответ:

Дистрибуция мобильных приложений — это предоставление возможности пользователям скачивать и обновлять приложения. Самый простой способ дистрибуции Android-приложения — это компиляция сборки и загрузка в Google Play. Сама дистрибуция осуществляется посредством загрузки APK или AAB файла приложения на платформу для распространения.

APK - формат архивных исполняемых файлов-приложений для Android. APK является стандартным форматом приложений в ОС Android, его установка проста и не требует дополнительных программ. В Android Studio APK файл приложения можно создать в разделе Build.

Также существует формат Android App Bundle – более универсальный формат, обеспечивающий работу приложения на разных устройствах (т.е. в то время как APK должны быть разных версий под разные устройства, AAB будет иметь одну, подходящую для всех). AAB содержит компоненты, позволяющие генерировать подходящую под устройство пользователя APK версию приложения. Минус AAB – более сложный процесс установки. Создать AAB формат тоже можно в разделе Build в Android Studio. С 2021 года формат является обязательным для всех новых приложений, опубликованных в Google Play.

Экосистема Android достаточно открыта и позволяет загружать приложения не только в официальный маркет от Google - Google Play. Есть и сторонние маркеты, например Amazon App Store, APKMirror и т.д. Также распространять Android-приложения можно и через веб-сайты, посредством простой загрузки APK/AAB файлов в открытый доступ.

# 5. Набор инструментов, необходимых для разработки приложения.

Ответ:

Для полноценной разработки приложений для мобильных устройств необходимо непосредственно мобильное устройство, подключенное к компьютеру, на котором пишется код, либо эмулятор мобильного устройства.

Для разработки на Android необходимо установить следующее ПО:

Java SDK - компилятор Java со стандартными библиотеками. Т.к. нативное приложение пишется на этом языке программирования.

Android SDK - содержит в себе эмулятор мобильного устройства, отладчик, набор библиотек.

IDE - например, Android Studio. Предоставляет текстовый и визуальный редактор, подсказки, быстрый доступ к документации, вставку boilerplate кода (шаблонный код).

# 6. Виды и типы мобильных приложений. Нативные, веб и гибридные приложения.

Ответ:

Виды приложений:

* Нативные. Нативные мобильные приложения — самый распространенный в разработке вид, так как создается отдельно для каждой ОС (iOS, Android или другие). Spotify, яндекс музыка и подобные приложения являются нативными.
* Веб приложение. Веб-приложения представляют собой адаптированные веб-сайты, которые открываются через браузеры. Пользователь не скачивает приложение и не хранит его на своем устройстве. Сложно выделить примеры, чтобы не ошибиться. Разные источники приводят в пример Google Maps, программы Microsoft Office.
* Гибридные приложения. Это веб-приложения, которые выглядят как нативные и имеют их признаки. В частности, иконки на рабочем столе мобильного устройства пользователя, хорошую производительность и возможность работать в автономном режиме. Среди гибридных приложений выделяют Uber, Evernote.

Типы приложений:

* Приложения переднего плана. Работают только когда видимы на экране (игры, видеопроигрыватели и тд)
* Фоновые приложения. Работают в фоне, не предполагают активное взаимодействие с пользователем после настройки (sms автоответчики)
* Смешанные приложения. Смешанная версия первых двух. Допускают взаимодействие после настройки пользователем (чаты и почта)
* Виджеты. Приложения в виде графического объекта на рабочем столе (отображение погоды)

# 7. Основные языки программирования, применяющиеся для разработки мобильных приложений.

Ответ:

* Kotlin
* Swift
* Java
* Objective С
* JavaScript

Kotlin: язык программирования для создания приложений под Android. Статически типизированный, объектно-ориентированный язык программирования, работающий поверх Java Virtual Machine и разрабатываемый компанией JetBrains. Также компилируется в JavaScript и в исполняемый код ряда платформ через инфраструктуру LLVM.

Swift: Открытый мультипарадигмальный компилируемый язык программирования общего назначения. Создан компанией Apple в первую очередь для разработчиков iOS и macOS. Swift не только перенял все возможности Objective С, но и был наделен новыми функциями, которые упрощают написание и реализацию кода.

Java: Строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems.

Objective С: Компилируемый объектно-ориентированный язык программирования, используемый корпорацией Apple, построенный на основе языка Си и парадигм Smalltalk. В 2014 году вышел более мощный Swift, который взял себе все лучшее от Objective-C, но был лишен его недочетов. Сейчас большинство программистов выбирают Swift, но Objective С все еще используется для поддержки Legacy проектов.

JavaScript: Мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. В контексте мобильных приложений JavaScript применяется в React Native: это кроссплатформенный фреймворк с открытым исходным кодом для разработки мобильных и настольных приложений. React Native поддерживает такие платформы как Android, iOS, macOS, Web, Windows и UWP, позволяя разработчикам использовать возможности библиотеки React вне браузера для создания приложений, имеющих полный доступ к системным API платформ.

# 8. Необходимое программное обеспечение для разработки нативных мобильных приложений.

Ответ:

Нативные приложения (англ. native application) — это прикладные программы, которые были разработаны для использования на определённой платформе или на определённом устройстве. Одно из преимуществ нативных приложений — то, что они оптимизированы под конкретные операционные системы, поэтому они могут работать корректно и быстро.

Нативное приложение специально создается и разрабатывается для конкретной мобильной платформы на ее родном языке программирования, а именно:

* iOS (Objective-C или Swift)
* Android (Java или Kotlin)

Рассмотрим ПО для нативной разработки каждой из платформ:

* Android Studio — официальная интегрированная среда разработки (IDE) для работы с платформой Android, является наиболее популярной среди разработчиков. С её помощью можно создавать продукты для смартфонов, планшетов, носимых устройств, телевизоров и других устройств. Работая с Android Studio, вы получаете не только мощный редактор кода, но и множество дополнительных функций, значительно расширяющих возможности разработчика. Список возможностей Android Studio расширяется с каждой новой версией. Вот некоторые из функций, доступных на данный момент:

1. гибкая система сборки, основанная на Gradle;
2. шаблоны основных макетов и компонентов Android;
3. интеграция с облачной платформой Google и App Engine;
4. широкий спектр фреймворков и инструментов тестирования;
5. поддержка C++ и собственного комплекта разработки (NDK);
6. функция предпросмотра макета на нескольких конфигурациях экрана;
7. статистический анализатор кода, который позволяет находить проблемы производительности, несовместимости версий;
8. многофункциональный эмулятор, обеспечивающий отличную производительность.

Ещё одним преимуществом можно назвать то, что в качестве IDE Android Studio позволяет создавать собственные приложения практически в любой операционной системе, будь то Windows, Mac или Linux. Для использования этой возможности необходимо установить комплект разработки Java (JDK). Помимо прочих инструментов разработки JDK включает среду выполнения Java (JRE), интерпретатор кода (Java) и компилятор (Javac).

С таким обширным набором функций популярность Android Studio не вызывает вопросов.

Отличными примерами нативных приложений, созданных с помощью Android Studio, являются WhatsApp, LinkedIn, Netflix, Evernote, Uber.

* XCode (iOS) — это официальная интегрированная среда разработки от Apple. Первая версия была выпущена в 2003 году, поддерживает не только Objective-C, Swift, но и C, C++, Java, Python, AppleScript и Ruby. Стабильные версии IDE распространяются бесплатно через Mac App Store.

В XCode есть практически всё необходимое для разработки приложений на iOS: компиляторы, отладчики, редакторы раскадровок и конструктор XML-интерфейсов (XIB). Некоторые функции создания приложений для iOS доступны только в XCode. Например, при использовании Flutter или React Native невозможно будет редактировать раскадровки без использования XCode.

Некоторые функции, доступные в XCode:

1. поддержка разработки приложений для компьютеров на процессорах архитектуры ARM;
2. фильтр кода, редактор версий, помощник по кодированию, система управления версиями и т.д.;
3. графический интерфейс пользователя (GUI), который облегчает разработку и прототипирование пользовательского интерфейса;
4. функция проверки исходного кода, работающая в режиме реального времени;
5. локальная тестовая среда — XCode поддерживает тестирование покупок в приложении.

Разрабатывая приложение с использованием XCode, вы облегчаете процесс отправки приложения в App Store.

Документация XCode содержит подробную справку, в том числе документацию SDK, ссылки на API, примеры кода, руководства по кодированию — практически всё, что необходимо разработчику для эффективного устранения неполадок.

https://punicapp.com/blog/pages/1593/krossplatformennaya\_i\_nativnaya\_mobilnaya\_razrabotka

<https://mopinion.com/mobile-app-development-tools-an-overview/>

<https://www.fabrizioduroni.it/2018/01/16/ide-refactoring-android-studio-xcode-appcode-webstorm-jetbrains/>

Транслятор - то что переводит код на Java в ByteCode. Android имеет gradle. Сборщик собирает .apk

# 9. Основные характеристики языка программирования Java.

Ответ:

Особенности языка:

Объектно-ориентированный язык общего назначения;

С-подобный синтаксис;

Разработан компанией Oracle ;

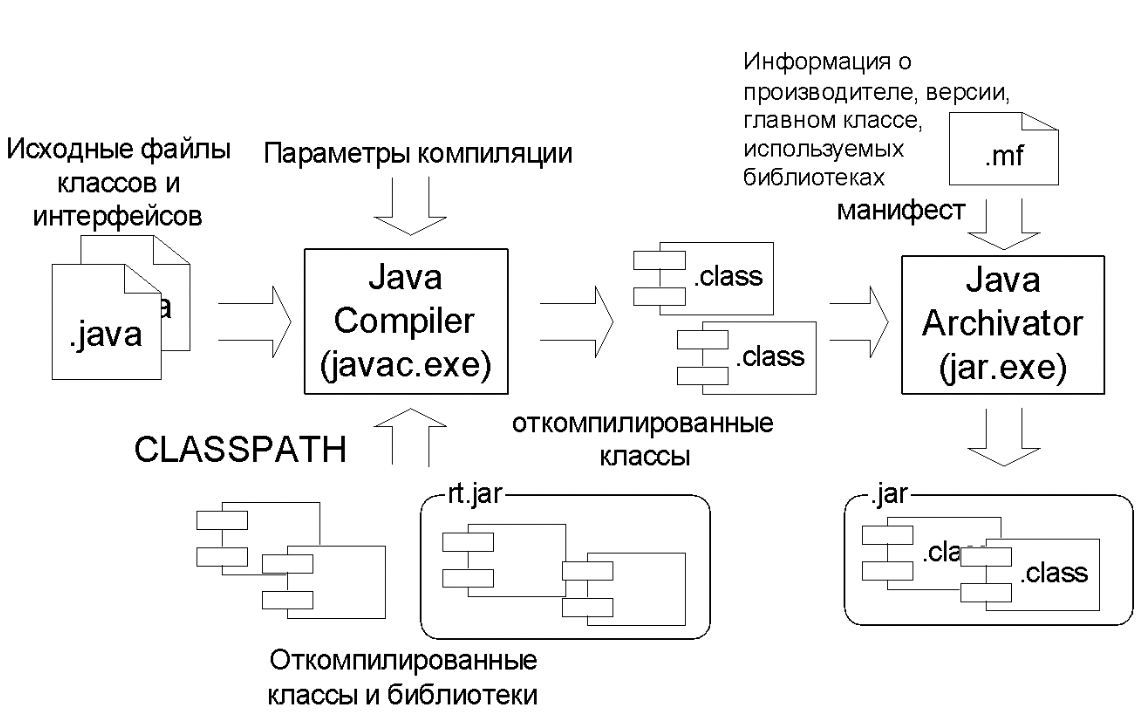
Существуют бесплатные инструменты разработки;

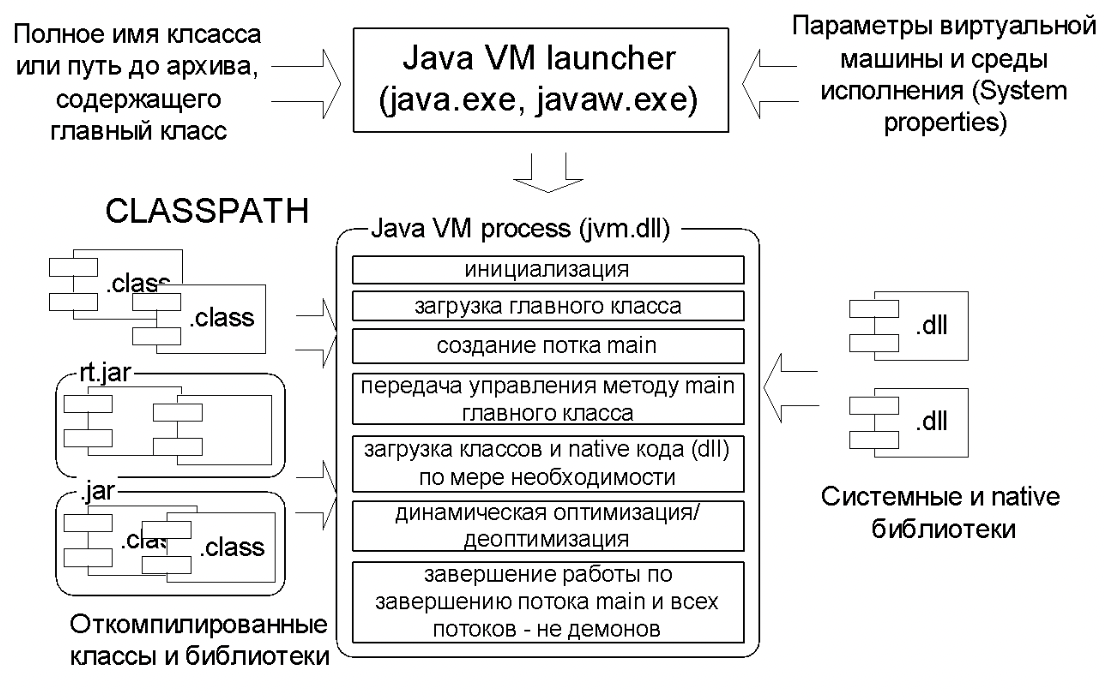
Компиляция в байт-код, использование нативного кода;

Виртуальные машины доступны для всех распространенных платформ;

Развитая модельная и объектно-ориентированная система;

Явная статическая сильная типизация;

Цикл разработки и компиляции:Запуск и исполнение:



# 10. Система типов в Java.

Ответ:

​​Переменные не что иное как зарезервированные места памяти для хранения значений. Это означает, что при создании переменной Вы резервируете некоторое пространство в памяти.

Каждая переменная и каждое выражение в Java обладает типом и этот тип строго определен.

Существует два типа данных в Java:

· простые или примитивные типы данных;

· ссылочные типы данных (ссылка/объект).

1) Есть восемь примитивных типов данных, поддерживаемых Java и названные по ключевому слову.

Тип byte

- Тип данных byte является 8-разрядным знаковым целым числом.

- Минимальная величина равна -128 (-2^7).

- Максимальное значение равно 127 (включительно) (2^7-1).

- По умолчанию – 0.

- byte предназначен для экономии места в больших массивах, главным образом вместо целых чисел, поскольку byte в четыре раза меньше, чем int.

Пример:

byte a = 100;

byte b = -50;

Тип short

- Тип данных short является 16-разрядным знаковым целым числом.

- Минимальное значение равно -32768 (-2^15).

- Максимальная величина равна 32 767 (включительно) (2^15-1).

- Тип short в Java может также использоваться для экономии памяти как byte.

- Размер short в 2 раза меньше, чем int.

- По умолчанию — 0.

Пример:

short s = 10000;

short r = -20000;

Тип int

- В языке Java тип данных int является 32-разрядным знаковым целым числом.

- Минимальный размер int — 2 147 483 648 (-2^31).

- Максимальная величина равна 2,147,483,647 (включительно)(2^31-1).

- Тип int обычно используется для целых значений. Если нет озабоченности по поводу памяти.

- По умолчанию равно 0.

Пример:

int a = 100000;

int b =-200000;

Тип long

- Тип данных long является 64-разрядным знаковым целым числом.

- Минимальное значение равно — -2^63.

- Максимальная величина — 2^63-1.

- В Java Применяется когда требуется более широкий диапазон, чем int.

- По умолчанию — 0L.

Пример:

long a = 100000L;

long b =-200000L;

Тип float

- Тип данных float является c одинарной точностью 32-битный IEEE 754 с плавающей точкой.

- Тип float используется главным образом для сохранения памяти в больших массивах чисел с плавающей точкой.

- По умолчанию — 0.0f.

- Тип float никогда не должен применяется для точного значения.

Пример:

float f1 = 234.5f;

Тип double

- Тип данных double является c двойной точностью 64-битный IEEE 754 с плавающей точкой.

- Обычно используется для десятичных значений.

- Тип double никогда не должен применяется для точного значения.

- По умолчанию — 0.0d.

Пример:

double d1 = 123.4;

Тип boolean

- Тип данных boolean представляет собой один бит информации.

- Существует только два возможных значения: true и false.

- Предназначен для простых признаков, которые позволяют отслеживать условия true или false.

- По умолчанию — false.

Пример:

boolean one = true;

Тип char

-Тип данных char является одним 16-разрядным символом Юникода.

- В Java char нужен для хранения любого символа.

Пример:

char letterA ='A';

2) Ссылочные типы данных

Пример:

Animal animal = new Animal("giraffe");

В ссылочные типы входят все классы, интерфейсы, массивы. Этот класс из стандартной библиотеки Java. Объекты класса и различные виды переменных массива подпадают под ссылочный тип данных. Ссылочная переменная может применяться для обозначения любого объекта, объявленного или любого совместимого типа.

По умолчанию в Java значение любой переменной ссылки - null.

Также существуют классы-оболочки:

· Byte

· Short

· Integer

· Long

· Float

· Double

· Character

· Boolean

В отличие от примитивных типов, они пишутся с заглавной буквы. Эти типы соответствуют примитивным типам, однако являются ссылочными. Их классы cодержат методы для преобразования типов, а также другие константы и методы полезные при работе с примитивными типами данных.

Тип String также относится к ссылочным типам и предназначен для хранения строк текста.

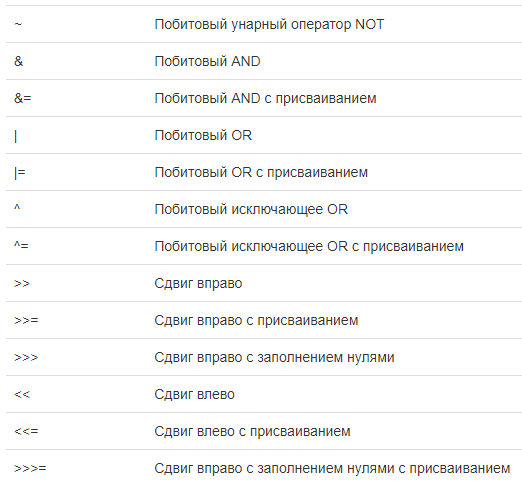
# 11. Основные операторы Java. Условные операторы, операторы ветвления, ввод-вывод.

Ответ:

Основные операторы языка:

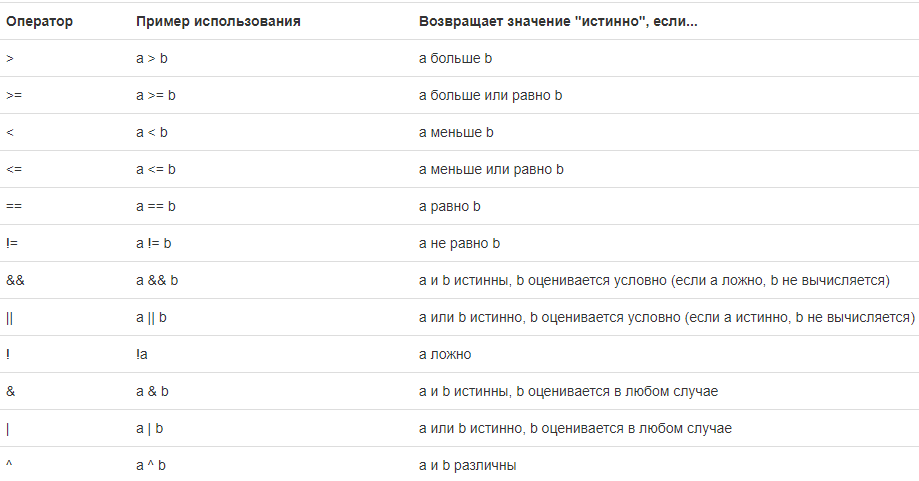
* Побитовые операторы

Существует несколько побитовых операторов, применимых к целочисленными типам long, int, short, char, byte.



* Операторы сравнения

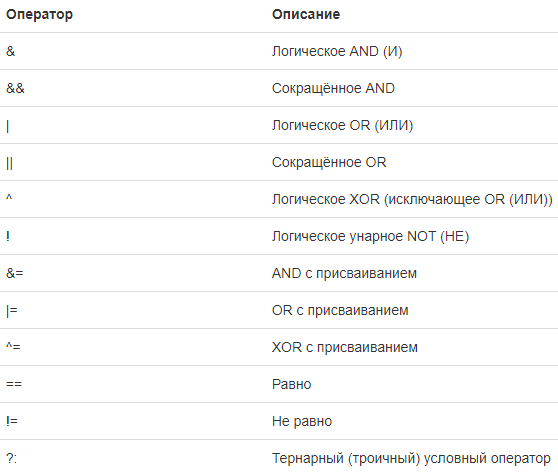
Операторы сравнения выдают логический результат (boolean). Если условие проверки истинно, оператор выдает true, а если ложно - false.



Операторы "Равно" и "не равно" можно использовать для всех примитивных типов данных, однако остальные сравнения нельзя использовать к типу boolean.

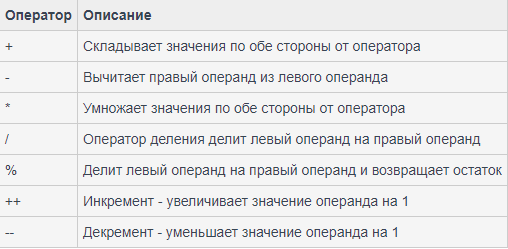
* Логические операторы

Логические операторы работают только с операндами типа boolean. Все логические операторы с двумя операндами объединяют два логических значения, образуя результат.



* Арифметические операторы

Арифметические операторы — используются в математических выражениях таким же образом, как они используются в алгебре.



Условный оператор:

Оператор if позволяет вашей программе в зависимости от условий выполнить оператор или группу операторов, основываясь на значении булевой переменной или выражения. Оператор if является основным оператором выбора в Java и позволяет выборочно изменять ход выполнения программы - и это одно из основных отличий между программированием и простым вычислением.

if (условие) {оператор}; // если условие истинно, то выполняется оператор

Список операторов, которые можно использовать в условных выражениях:



Существует расширенный вариант оператора if с использованием ключевого слова else:(оператор ветвления)

if (условие) {оператор}; // если условие истинно, то выполняется первый оператор

else {оператор}; // если условие ложно, то выполняется оператор

после else

Также существует цепочка операторов if-else-if - конструкция, состоящая из вложенных операторов if:

if (условие)

{оператор};

else if (условие)

{оператор};

else if (условие)

{оператор};

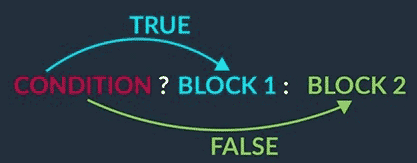
else

{оператор};

Условные выражения оцениваются сверху вниз. Как только найдено условие, принимающее значение true, выполняется ассоциированный с этим условием оператор, а остальная часть цепочки пропускается. Если ни одно из условий не принимает значение true, то выполняется последний оператор else, который можно рассматривать как оператор по умолчанию. Если же последний оператор else отсутствует, а все условные выражения принимают значение false, то программа не выполняет никаких действий.

Также можно использовать ? : вместо if-else

логическое\_условие ? выражение1 : выражение2



**Оператор switch**

Команду switch часто называют командой выбора. Выбор осуществляется в зависимости от целочисленного выражения. Форма команды выглядит так:

switch(ВыражениеДляСравнения) {

case Совпадение1:

команда;

break;

case Совпадение2:

команда;

break;

case Совпадение3:

команда;

break;

default:

оператор;

break;

}

**Операторы перехода break, continue, return**

Используя оператор break, можно вызвать немедленное завершение цикла, пропуская условное выражение и другой код в теле цикла. Когда приложение встречает оператор break внутри цикла, оно прекращает выполнение цикла и передает управление оператору, следующему за циклом.

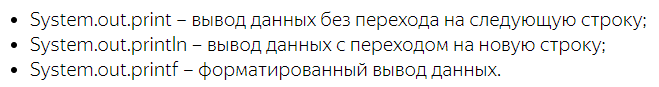
Оператор return используют для выполнения явного выхода из метода. Оператор можно использовать в любом месте метода для возврата управления тому объекту, который вызвал данный метод. Таким образом, return прекращает выполнение метода, в котором он находится.

Иногда полезно начать очередную итерацию цикла пораньше. То есть нужно продолжить выполнение цикла, но прекратить обработку остатка кода в его теле для данной частной итерации. Такое действие выполняет оператор continue. В циклах while и do while оператор continue вызывает передачу управления непосредственно условному выражению, которое управляет циклом. В цикле for управление переходит сначала к итерационной части оператора for и затем к условному выражению.

Ввод и вывод:

Чтобы обеспечивать взаимодействие с консолью, в языке программирования Java используют класс System.

Чтобы создать потока вывода в вышеупомянутый класс System, вам понадобится специальный объект out. В нем определен метод println, обеспечивающий вывод значения на консоль и перевод курсора консоли на другую строку.



public class Main {

public static void main(String[] args) {

System.out.println("Привет, мир!");

}

}

Чтобы обеспечить ввод с консоли Java, в классе System есть объект in. Именно через объект System.in работать не очень удобно, поэтому часто применяют класс Scanner. Он уже, в свою очередь, как раз таки и применяет System.in

**import** java.util.Scanner**;**

**public** **class** **Main** **{**

**public** **static** **void** **main(**String**[]** args**)** **{**

Scanner in **=** **new** Scanner**(**System**.**in**);**

System**.**out**.**print**(**"Введите любой номер: "**);**

**int** num **=** in**.**nextInt**();**

System**.**out**.**printf**(**"Ваш номер: %d \n"**,** num**);**

in**.**close**();**

**}**

**}**

# 12. Функции в Java. Передача аргументов.

Ответ:

Функции в Java именуют методами, поскольку они определены в классе. Метод - это группа команд для многократного вызова, у которой есть уникальное имя.

Синтаксис:

public static void имя(набор параметров)

{

код метода

}

, где

имя - уникальное в рамках класса имя, не может начинаться с цифры

код метода - функционал, который реализует метод

void - тип возвращаемого значения (может быть например int, double, String)

static - модификатор, указывающий на то, что метод принадлежит классу (для использования не нужно создавать объект), опционален

public - модификатор доступа (может быть например default - по умолчанию, protected, private), указывающий в данном примере на то, что метод может быть использован извне

набор параметров - описание параметров, которые принимает метод, опционален.

Параметры - это переменные, которые используются при вызове/создании метода, аргументы - это фактические значения (данные), которые передаются методу при вызове. В Java аргументы **всегда** передаются по значению. То есть в случае примитивных типов (int, double, char и др. ) в функцию передается не ссылка, а копируется само значение. Но при передачи ссылок на объект передается уже не копия объекта, а содержимое ссылки - средство управления / доступа к объекту.

# 13. Реализация ООП в Java. Наследование, инкапсуляция, полиморфизм. Методы и атрибуты класса.

Ответ:

Наследование — еще одна важная концепция ООП, которая позволяет сэкономить время на написании кода. Возможности наследования раскрываются в том, что новому классу передаются свойства и методы уже описанного ранее класса. Класс, который наследуется называется дочерним (или подклассом). Класс, от которого наследуется новый класс — называется родительским, предком и т. д. В языке программирования Java используется ключевое слово **extends** для того, чтобы указать на класс-предок.

Инкапсуляция позволяет нам пользоваться возможностями класса без создания угрозы безопасности данных за счет ограничения прямого доступа к его полям. Также она позволяет изменять код классов не создавая проблем их пользователям (другим классам). В Java данный принцип достигается за счет использования ключевого слова **private**.

Полиморфизм предоставляет возможность единообразно обрабатывать объекты с различной реализацией при условии наличия у них общего интерфейса или класса. По-простому: способность вызывать нужные методы у объектов, имеющие разные типы (но находящиеся в одной иерархии). При этом происходит автоматический выбор нужного метода в зависимости от типа объекта.

Атрибуты - Переменные, созданные в классах **Java**, называются **атрибуты** или поля класса. **Атрибуты** класса к ним можно получить доступ с помощью объекта класса и с использованием синтаксиса точки (.).

# 14. Структура проекта приложения для Android. Основные категории файлов.

Ответ:

Проект в Android Studio представлен 2 корневыми папками: **app** и **Gradle Scripts.** В **app** входят папки:

·**manifests** (файлы конфигураций, файлы манифеста). Файл AndroidManifest.xml содержит всю необходимую информацию, используемую системой для запуска и выполнения приложения. Основная информация в манифесте: 1) Имя Java пакета приложения; 2) Описание компонентов приложения; 3) Определение процессов; 4) Объявление полномочий, которыми должно обладать приложение для доступа к защищенным частям API и взаимодействия с другими приложениями; 5) Объявление полномочий, которыми должны обладать другие приложения для взаимодействия с компонентами данного; 6) Список вспомогательных классов; 7) Определение мин.уровня Android API для приложения; 8) Список библиотек, связанных с приложением.

· **java.** Папка java содержит исходный код приложения. Классы могут быть расположены в различных пакетах, но обязательно внутри папки java. По умолчанию папка включает файл MainActivity, содержащий код на Java, который представляет класс MainActivity, запускаемый по умолчанию при старте приложения.

· **res.** Содержит файлы ресурсов, например изображения, xml файлы, анимации, звуковые файлы и т.д. Внутри папки res ресурсы распределены по своим папкам: 1) Папка drawable содержит файлы с изображениями; 2) Папка layout содержит xml файлы для построения пользовательского интерфейса; 3) mipmap содержит значки/иконки приложения; 4) values содержит xml файлы, в которых определяются простые значения типа строк, массивов, целых чисел, размерностей, цветов и стилей; (эти 4 подкатегории генерируются при создании проекта) 5) В папке menu находятся xml файлы, используемые только для создания меню; 6) animator содержит свойства анимации; 7) anim содержит анимации преобразований; 8) color содержит списки цветов; 9) raw содержит различные файлы, которые сохраняются в исходном виде; 10) font включает файлы с определениями шрифтом и расширениями .ttf, .otf или .ttc, либо файлы XML, которые содержат элемент <font-family>.

**Gradle Scripts** включает скрипты, которые 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.

# 15. Файл манифеста. Назначение, основные разделы

Ответ:

<http://developer.alexanderklimov.ru/android/theory/AndroidManifestXML.php>

Файл манифеста AndroidManifest.xml **должен** быть в каждом Android приложении.

В этом файле **описываются** компоненты приложения, содержит список необходимых разрешений для обращения к данным с устройства и защищенным частям API.

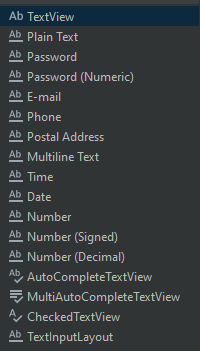
Файл манифеста **содержит** разрешения для взаимодействия с другими приложениями: можно указать разрешения, которые обязано иметь другое приложения, для взаимодействия с компонентами данного приложения.

**Системная информация**: имя Java-пакета (служит уникальным идентификатором), связанные библиотеки, необходимый минимальный уровень Android API

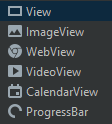
# 16. Элементы пользовательского интерфейса Android SDK: Текстовые, графические, элементы управления формами, кнопки. Способы создания элементов.

Ответ:

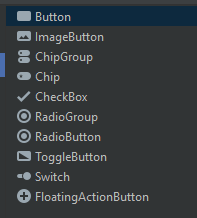
Текстовые элементы. В частности нас интересуют основные элементы, такие как TextView для отображения обычного текста и EditText для ввода информации, но в Android Studio существует множество текстовых элементов:



Графические элементы. Из графических элементов нас в основном интересует ImageView для отображения картинок. Есть еще несколько графических элементов:



Кнопки. Обычные кнопки и кнопки с картинкой также могут быть использованы в форме. Элементы управления формами т.е. чекбоксы, радиогруппы, радиокнопки и многие другие элементы управления формой можно также увидеть в Android Studio:



Элементы задаются в XML файле. Пример кода ниже создает текстовое окошко и кнопку.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<TextView android:id="@+id/text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, I am a TextView" />

<Button android:id="@+id/button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, I am a Button" />

</LinearLayout>

# 17. Принцип разделения представления и поведения. Файлы представлений, ресурсы, программный код.

Ответ:

Разделение поведения и представления (или разделение контента и стиля) - это принцип разделения дизайна применительно к созданию и представлению контента. В соответствии с этим принципом визуальные и дизайнерские аспекты (презентация и стиль) отделены от основного материала и структуры (содержания) документа. Типичной аналогией, используемой для объяснения этого принципа, является различие между человеческим скелетом (как структурным компонентом) и человеческой плотью (как визуальный компонент), который составляет внешний вид тела. Общие применения этого принципа можно увидеть в веб-дизайне (HTML против CSS) и верстке документов (текст документа Ламберта против его преамбулы).

(может уместно применить разделение к java и xml)

Этот принцип не является жестким руководством, а скорее служит наилучшей практикой для разделения внешнего вида и структуры. Во многих случаях аспекты проектирования и разработки проекта выполняются разными людьми, поэтому разделение обоих аспектов обеспечивает как первоначальную ответственность за производство, так и последующее упрощение обслуживания, как в случае с принципом "Не повторяйся" (DRY).

**Пример кода**:

С xml/java:

<EditText

android:id="@+id/editText"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginTop="29dp"

android:ems="10"

android:inputType="textPersonName"

android:text="Name"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

private EditText mNameEditText;

mNameEditText = findViewById(R.id.editText);

public void onClick(View view) {

if (mNameEditText.getText().length() == 0) {

mHelloTextView.setText("Hellо!");

} else {

mHelloTextView.setText("Привет, " + mNameEditText.getText());

}

}

**Ресурсы**

**Представление** **ресурса** — **это** **обычно** **документ**, **отражающий** **текущее** **или** **требуемое** **состояние** **ресурса**. **Ресурсы** **обычно** представляются **документами** в форматах XML или JSON. JSON— эффективный текстовый формат кодирования данных, обеспечивающий быстрый обмен небольшими объемами данных между клиентскими браузерами и веб-службами с поддержкой AJAX.

**Файлы представлений**

Представления отвечают за вывод данных возвращаемых запросом в определенной форме. Часто это происходит в виде HTML, XML или JSON

# 18. Компоненты Android приложения. Виды, назначение. Способ описания. Регистрация компонентов в манифесте. Жизненный цикл компонентов.

Ответ:

Компоненты Android приложения:

* Activity/Деятельность. Activity представляет собой схему представления Android-приложений. Например, экран, который видит пользователь. Android-приложение может иметь несколько активити и может переключаться между ними во время выполнения приложения. Пользовательский интерфейс нашего приложения построен вокруг одного или нескольких расширений класса Activity. Используя фрагменты и представления, activities устанавливают макет и отображают выходные данные, а также реагируют на действия пользователя. Действие реализуется как подкласс класса Activity.
* Service/Сервис. Сервисы подобны невидимым работникам нашего приложения. Эти компоненты работают на бэкэнде, обновляя источники данных и действия, вызывая уведомления, а также широковещательные намерения. Они также выполняют некоторые задачи, когда приложения не активны. Сервис может использоваться как подкласс класса Service
* Content Provider/Контент провайдер. Он используется для управления и сохранения данных приложения, а также обычно взаимодействует с базой данных SQL. Они также отвечают за обмен данными за пределами границ приложения. Контент-провайдеры конкретного приложения могут быть настроены так, чтобы разрешить доступ из других приложений, и контент-провайдеры, предоставляемые другими приложениями, также могут быть настроены.Поставщик контента должен быть подклассом класса ContentProvider.
* Broadcast Receiver/Широковещательный приемник. Они известны как слушатели намерений, поскольку они позволяют вашему приложению прослушивать намерения, которые удовлетворяют указанным нами критериям соответствия. Широковещательные приемники заставляют наше приложение реагировать на любое полученное намерение, что делает их идеальными для создания приложений, управляемых событиями.

# Каждый проект в Android включает в себя файл манифеста, который AndroidManifest.xml , хранится в корневом каталоге иерархии проекта. Файл манифеста является важной частью нашего приложения, поскольку он определяет структуру и метаданные нашего приложения, его компоненты и его требования. Этот файл включает в себя узлы для каждого из компонентов, которые создают приложение, и использование фильтров намерений и разрешения (Permission) которые определяют, как они координируются друг с другом и другими приложениями.

**Жизненный цикл** приложения в Android жёстко контролируется системой и зависит от нужд пользователя, доступных ресурсов и т. д. Например, пользователь хочет запустить браузер. Решение о запуске приложения принимает система. Хотя последнее слово и остаётся за системой, она подчиняется определённым заданным и логическим правилам, позволяющим определить, можно ли загрузить, приостановить приложение или прекратить его работу. Если в данный момент пользователь работает с определённым окном, система даёт приоритет соответствующему приложению. И наоборот, если окно невидимо и система решает, что работу приложения необходимо остановить, чтобы мобилизовать дополнительные ресурсы, будет прекращена работа приложения, имеющего более низкий приоритет. В Android ресурсы более ограниченны, поэтому Android более жёстко контролирует работу приложений.

Основные методы жизненного цикла приложения

* protected void onCreate()
* protected void onStart()
* protected void onRestart()
* protected void onResume()
* protected void onPause()
* protected void onStop()
* protected void onDestroy()

У методов onCreate(), onStart(), onResume() вызов суперкласса должен происходить до вызова вашего кода. В методах onPause(), onStop(), onDestroy() суперкласс следует вызывать после вашего кода

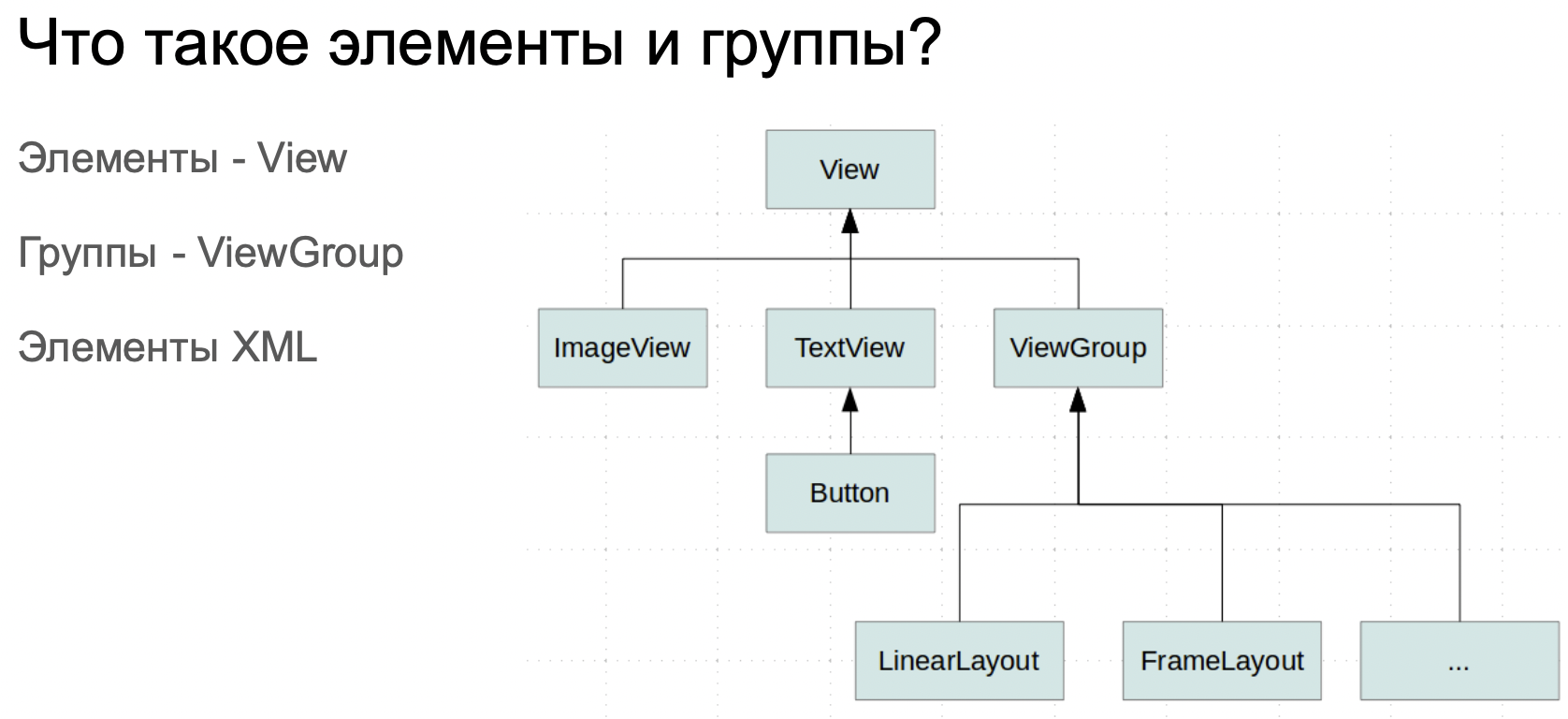
# 

# 19. Расположения элементов. Основные виды, особенности, основные атрибуты.

Ответ:

<https://www.youtube.com/watch?v=wj1ASnd27SQ>

Элементы в Android - это основные визуальные компоненты приложения. Их описание хранится в xml файлах в папке /res/layout.



Основные виды элементов:

Текстовые элементы, Графические, Кнопки

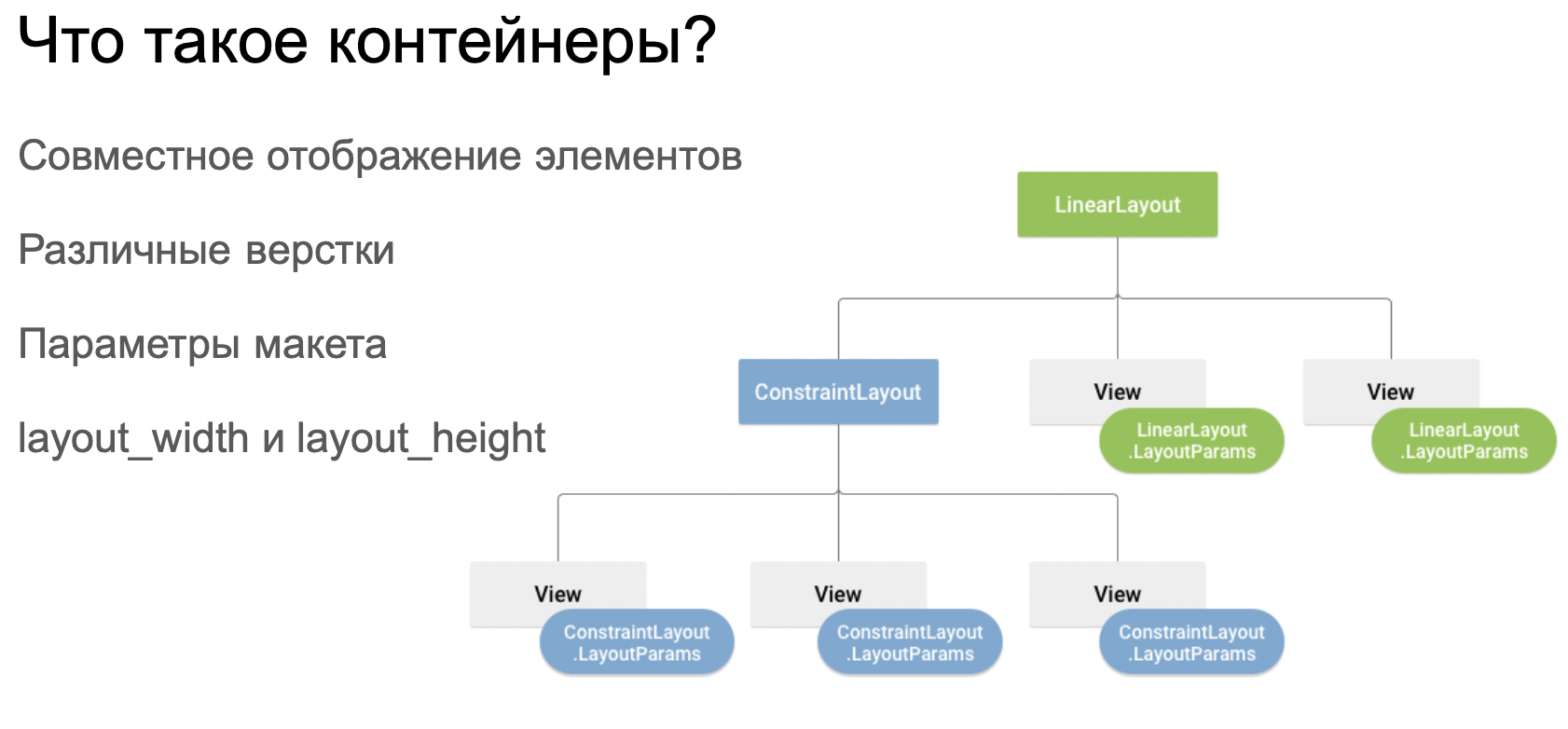
Основные виды расположения элементов:

View, Layout

Пример элементов Android приложения, которые использовались в ходе изучения курса: TextView, EditText, ImageView, Button, CheckBox и др.

Сами элементы, для грамотного позиционирования в Android приложении принято хранить в контейнерах(Layout’ах).

Контейнеры позволяют группировать элементы, совместно отображая элементы приложения. Этим позволяет реализовать полноценную верстку приложения.



Пример контейнеров Android приложения, которые использовались в ходе изучения курса: LinearLayout, TableLayout, FrameLayout, RelativeLayout, ConstraintLayout и др.

Основные атрибуты:

layout\_width - задание ширины, layout\_height -задание высоты, layout\_weight - пропорция элемента по горизонтали и вертикали, layout\_gravity - выравнивание элемента (left, center, right),

layout\_margin - отступ.

Для определения id элемента используется атрибут id=’name’

Для определения контекста контейнера используется атрибут context=’name\_of\_activity’

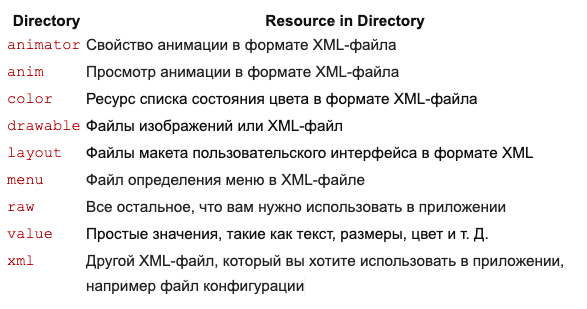
При позиционировании элементов используются единицы измерения Android: mm - миллиметры, in - дюймы, px - пиксели, pt - точки - 1/72 дюйма, dp - независимые пиксели, sp - пиксели, приведенные к размеру шрифта.

# 20. Ресурсы Android приложения. Виды, назначение. Организация доступа к ресурсам. Альтернативные ресурсы.

Ответ:

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.

Виды ресурсов и название их директорий:



​​По умолчанию в проекте каталоги существуют не для всех типов ресурсов, которые используются в Android, однако при необходимости мы можем добавить в папку res нужный каталог, а в него затем поместить ресурс.

Существует два способа доступа к ресурсам: в файле исходного кода и в файле xml.

1) Ссылка на ресурсы в коде

Тип ресурса в данной записи ссылается на одно из пространств (вложенных классов), определенных в файле R.java, которые имеют соответствующие им типы в xml:

* R.drawable (ему соответствует тип в xml drawable)
* R.id (id)
* R.layout (layout)
* R.string (string)
* R.attr (attr)
* R.plural (plurals)
* R.array (string-array)

Например, для установки ресурса activity\_main.xml в качестве графического интерфейса в коде MainActivity в методе onCreate() есть такая строка:

setContentView(R.layout.activity\_main);

Через выражение R.layout.activity\_main мы и ссылаемся на ресурс activity\_main.xml, где layout - тип ресурса, а activity\_main - имя ресурса.

2) Доступ в файле 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, где

* 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).

Например, мы хотим вывести в элемент TextView строку, которая определена в виде ресурса в файле strings.xml:

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/app\_name"

/>

В данном случае свойство text в качестве значения будет получать значение строкового ресурса app\_name.

Для получения ресурсов в классе Activity мы можем использовать метод getResources(), который возвращает объект android.content.res.Resources. Но чтобы получить сам ресурс, нам надо у полученного объекта Resources вызвать один из методов. Некоторые из его методов:

* getString(): возвращает строку из файла strings.xml по числовому идентификатору
* getDrawable(): возвращает графический файл в виде объекта Drawable
* getBoolean(): возвращает значение boolean

Альтернативные ресурсы

Большинство приложений предоставляет альтернативные ресурсы для поддержки определённых конфигураций устройств. Альтернативные ресурсы — это ресурсы, предназначенные для определенного устройства или конфигурации среды выполнения, например текущий язык, определенный размер экрана или плотность пикселей. Во время запуска приложения Android определит текущую конфигурацию устройства и загрузит соответствующие ресурсы.

Если для какой-либо конфигурации устройства нет доступных ресурсов, Android воспользуется ресурсами приложения по умолчанию.

Как и ресурсы по умолчанию, альтернативные ресурсы хранятся в папках внутри **res**. Для этих папок используется следующий шаблон названия:

<resource\_name>-<config\_qualifier>, где

* <resource\_name> имя папки для типа ресурса, например, **drawable** или **values**
* <config\_qualifier>определяет конфигурацию устройства, для которого будут использоваться ресурсы.

Чтобы добавить несколько квалификаторов к одному имени папки, нужно разделить их с помощью тире. Например, строковые ресурсы, локализованные на японский язык, будут находиться в файле strings.xml внутри папки values-ja в папке res (сокращенно res/values-ja/strings.xml)

Чтобы создать директорию альтернативных ресурсов, необходимо при создании ресурсной директории выбрать необходимые тип и квалификаторы в диалоговом окне New Resource Directory. Альтернативные файлы ресурсов должны быть названы точно так же, как файлы ресурсов по умолчанию (например, styles.xml или dimens.xml).

# 21. Активности (активити). Назначение, создание. Методы жизненного цикла.

Ответ:

Activity – основная единица графического интерфейса (аналог окна или экранной формы). Его основное предназначение заключается в том, что оно служит точкой входа для взаимодействия приложения с пользователем, а также отвечает за то, как пользователь перемещается внутри приложения или между приложениями. По сути активити - это окно, с которым пользователь взаимодействует.

В Android Studio Activity создается автоматически при создании проекта. Но если нужно добавить еще одну активность, то выбираем папку в которой нужно создать, после чего левой клавишей мышки жмем и выбираем New → Activity → Empty Activity. После чего будет создана новая Activity. И в манифесте автоматически появится запись о созданном манифесте.

*<activity android:name=".SecondActivity"></activity>*

Жизненный цикл активити состоит из шести основных колбэков, вызываемых системой. В порядке вызова: onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy().

* onCreate() вызывается когда активити создается. В onCreate() вы должны вызвать метод setContentView().
* onStart() вызывается когда активити отрисована и видима пользователю.
* onResume() вызывается перед тем как активити станет доступна для взаимодействия с пользователем.
* onPause() – метод симметричный onResume(). Пользователь больше не может взаимодействовать с активити, но активити частично видна пользователю. В этом состоянии UI активити может изменяться.
* onStop() – метод симметричный onStart(). Вызывается, когда активити больше не видна пользователю.
* onDestroy() – метод симметричный onCreate(). Вызывается перед тем, как активити будет уничтожена системой.

# 22. Сервисы. Назначение, создание. Методы жизненного цикла.

Ответ:

Сервисы - такой тип компонентов Android-приложения, который используется для выполнения длительных операций и доступа к удалённым ресурсам. У сервисов нет пользовательского интерфейса, нет возможности к нему обращаться, сервис может продолжить работу, даже если породившая его активность уничтожена . Решают задачи по выполнению различных операций без прерываний независимо от жизненного цикла активности.

Примеры таких операций (для реализации в качестве сервиса) - фоновое воспроизведение музыки, синхронизация с удаленной базой данных, скачивание файлов по сети.

Виды сервисов: запущенные (started), независимые, и связанные (bound), прерываются при отключении от них последнего компонента, есть гибридный режим.

При создании сервиса в Android Studio файл манифеста автоматически дополняется новыми настройками, которые можно изменять в зависимости от функционала.

Создание сервиса в гибридном режиме:

public class MyService extends Service {

@Override

public void onCreate() { }

@Override

public IBinder onBind(Intent intent) { }

@Override

public IBinder onUnbind(Intent intent) { }

@Override

public int onStartCommand(Intent intent…) { }

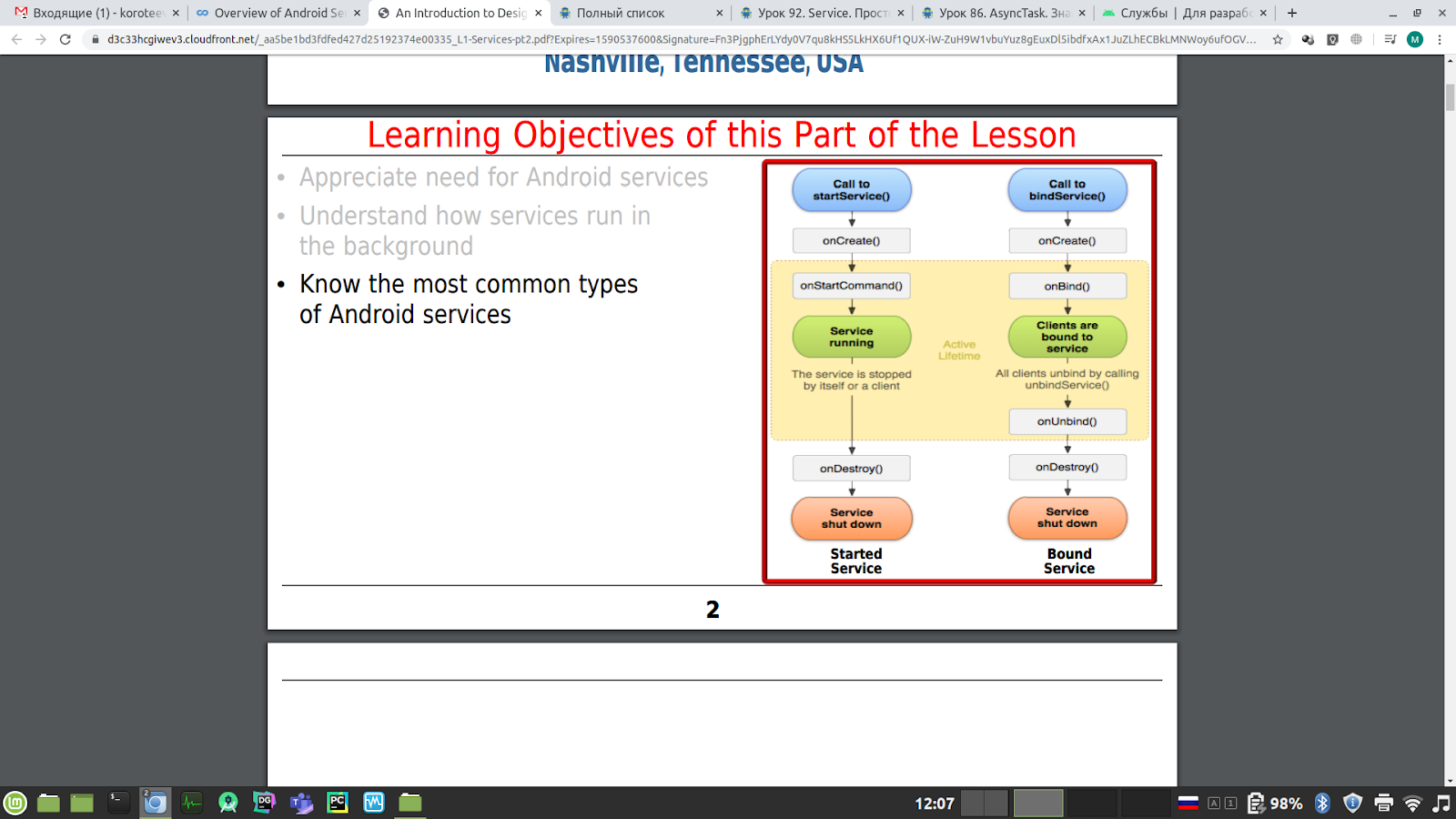
@Override

public void onDestroy() { }

}

Для реализации запущенного сервиса не нужно реализовывать методы onBind и onUnbind, для реализации связанного - не нужно реализовывать метод onStartCommand

Методы жизненного цикла должны определяться разработчиком, но явно их вызывает сама система, они являются callback-ами. Методы жизненного цикла имеют различия в зависимости от типа сервиса (их реализация описана выше). Порядок их запуска наглядно демонстрируется блок-схемами:



# 23. Поставщики (контент-провайдеры). Назначение, создание. Методы жизненного цикла.

Ответ:

<http://developer.alexanderklimov.ru/android/theory/contentprovider.php>

Контент-провайдер или "Поставщик содержимого" (Content Provider) - это оболочка (wrapper), в которую заключены данные. Если ваше приложение использует базу данных SQLite, то только ваше приложение имеет к ней доступ. Но бывают ситуации, когда данные желательно сделать общими. Простой пример - ваши контакты из телефонной книги тоже содержатся в базе данных, но вы хотите иметь доступ к данным, чтобы ваше приложение тоже могло выводить список контактов. Так как вы не имеете доступа к базе данных чужого приложения, был придуман специальный механизм, позволяющий делиться своими данными всем желающим.

Для создания собственного контент-провайдера нужно унаследоваться от абстрактного класса **ContentProvider**:

public class MyContentProvider extends ContentProvider {

}

В классе необходимо реализовать абстрактные методы (по сути жизненного цикла) **query()**, **insert()**, **update()**, **delete()**, **getType()**, **onCreate()**. Прослеживается некоторое сходство с созданием обычной базы данных.

А также его следует зарегистрировать в манифесте с помощью тега **provider** с атрибутами **name** и **authorities**. Тег **authorities** служит для описания базового пути URI, по которому **ContentResolver** может найти базу данных для взаимодействия. Данный тег должен быть уникальным, поэтому рекомендуется использовать имя вашего пакета, чтобы не произошло путаницы с другими приложениями, например:

<provider

android:name:".MyContentProvider"

android:authorities="ru.alexanderklimov.provider.notepad" />

# 24. Получатели широковещательных сообщений. Назначение, создание. Методы жизненного цикла.

Ответ:

Приемник широковещательных сообщений – особый тип компонента, который может реагировать на широковещательные сообщения. Приёмник представляет собой обычный Java-класс на основе BroadcastReceiver.

BroadcastReceiver – базовый класс для кода, который получает и обрабатывает рассылки (сообщения, упакованные в Intent и отправленные методом sendBroadcast(Intent)). Примерами таких сообщений могут быть: низкий заряд батареи, нажатие на кнопку камеры, установка нового приложения и т.д.

Для функционирования приёмника надо, во-первых, зарегистрировать его для получения конкретных интентов. Во-вторых, какой-либо компонент должен генерировать интент и передавать его в систему. В-третьих, Android должен доставлять этот интент получателям широковещательных сообщений, которые зарегистрированы для его получения. И четвертое в приемниках затем должен происходить вызов метода onReceive, в котором будет обработано входящее событие.

Приёмник можно зарегистрировать двумя способами:

1) Статическая регистрация в манифесте. Приёмник, заданный в манифесте, известен системе, которая сканирует файлы манифеста всех установленных приложений, т.е. даже если приложение не запущено, оно всё равно сможет отреагировать на поступающее сообщение Для статической регистрации в файле манифеста в секции application следует создать секцию receiver и указать класс приемника. Атрибут android:exported="true" сообщает получателю, что он может принимать широковещательные сообщения вне области приложения. Внутри секции указывается фильтр намерений в виде строки, чтобы определить, какие сообщения приёмник должен прослушивать.

<receiver

android:name=".StaticBroadcastReciever"

android:enabled="true"

android:exported="true" >

<intent-filter>

<action android:name="MY\_BROADCAST\_STATIC" />

</intent-filter>

</receiver>

(MainActivity.java, необязательно, добавила интент для полноценного примера)

public void sendUsingStaticallyRegisteredBroadcastReceiver(View view) {

Intent i = new Intent();

i.setAction("MY\_BROADCAST\_STATIC");

sendBroadcast(i);

}

2) Динамическая регистрация через registerReciever(). Приемник, созданный программно, может работать только в том случае, когда активность приложения активна. Такой метод используется, например, если приложение не запущено и ему нет смысла принимать сообщения о заряде батареи. Иначе заряд батареи устройства будет расходоваться ещё быстрее при лишней работе. Динамическая регистрация происходит с помощью метода Context.registerReceiver(). Для отмены регистрации используется метод unregisterReceiver().

MainActivity.java

DynamicBroadcastReceiver dynamicBroadcastReceiver = new DynamicBroadcastReceiver();

public void registerBroadcastReceiverDynamically(View view) {

IntentFilter filter = new IntentFilter();

filter.addAction("MY\_BROADCAST");

registerReceiver(dynamicBroadcastReceiver, filter);

}

(Необязательно, интент для полноценного примера

public void sendUsingDynamicallyRegisteredBroadcastReceiver(View view) {

Intent i = new Intent();

i.setAction("MY\_BROADCAST");

sendBroadcast(i);

})

@Override

protected void onDestroy() {

if (dynamicBroadcastReceiver != null) {

unregisterReceiver(dynamicBroadcastReceiver);

}

super.onDestroy();

}

Для реализации широковещательного приемника создается класс, расширяющий базовый класс BroadcastReceiver, где реализуется метод onReceive().

public class myBroadcastReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

(Необязательно, можно не конкретизировать действие и оставить “...”

Toast.makeText(context, "Broadcast", Toast.LENGTH\_SHORT).show();)

}

}

Приемник широковещательных сообщений имеет единственный метод жизненного цикла: onReceive(). Когда широковещательное сообщение прибывает для получателя сообщения, Android вызывает его методом onReceive() и передаёт в него объект Intent, содержащий сообщение. Приемник широковещательных сообщений является активным только во время выполнения этого метода. Когда программа возвращается из метода onReceive(), приёмник становится неактивным и система полагает, что работа объекта BroadcastReceiver закончена.

# 25. Приложение, состоящее из нескольких активностей. Взаимодействие между ними. Фильтры намерений.

Ответ:

Приложение, состоящее из нескольких активностей, имеет **бэкстек**. В нём содержится стек активностей, где текущая активность является последней. При нажатии на кнопку “Назад” текущая активность уничтожается, открывается последняя в стеке активность.

Активность позволяет запустить другую активность с помощью намерения. Сделать это можно в явном виде, указав класс активности, которую требуется вызвать, либо неявно, указав параметры action, data, category.

**Фильтр намерени**й описывает способность компонента получать набор намерений. Это необходимо при **неявном** вызове активности с помощью намерения, в котором указаны параметры **action**, **data**, **category**. Таким образом, у активности задаётся фильтр намерений, с помощью которого при неявном вызове активности система ищет среди всех активностей всех приложений те, чей фильтр намерений намерений имеет параметры, идентичные указанным в неявном вызове активности. Если нашлась одна активность, то открывается она. Иначе пользователю предоставляется выбор из приложений, в которых найдены подходящие активности.

# 26. Явные и неявные фильтры намерений. Параметры. Действия. Взаимодействие с системными активностями.

Ответ:

Намерение (Intent) - это механизм для описания одной операции - выбрать фотографию, отправить письмо, сделать звонок, запустить браузер и перейти по указанному адресу.

Наиболее распространенный сценарий использования намерения - запуск другой активности в своём приложении.

Явный интент:

Intent intent = new Intent(HelloWorld.this, AboutActivity.class);

startActivity(intent);

В намерении мы явно указали имя класса активности во втором параметре. Данный класс у нас прописан в манифесте, и система, просматривая манифест, запустит нужную активность.

Неявный интент:

Существует также неявный вызов активности. В этом случае при создании намерения мы используем не имя класса, а указываем параметры action, data, category с определёнными значениями. Система анализирует полученные данные и начинает подбирать подходящий вариант для запуска. Например, если были указаны в параметрах адрес веб-страницы, то скорее всего хотим запустить браузер.

Простой пример создания намерения с предопределенным действием ACTION\_VIEW для запуска браузера и перехода на нужный адрес:

Uri address = Uri.parse("[https://www.google.ru](https://www.google.ru/)");

Intent openLinkIntent = new Intent(Intent.ACTION\_VIEW, address);

if (openLinkIntent.resolveActivity(getPackageManager()) != null) {

startActivity(openlinkIntent);

} else {

Log.d("Intent", "Не получается обработать намерение!");

}

В данном случае действие ACTION\_VIEW означает просмотр чего-либо. Мы указали нужные данные (адрес), и происходит запуск новой активности (браузера). При этом исходная активность приостанавливается и переходит в фоновый режим.

Основные действия, которые надо знать:

ACTION\_VIEW

ACTION\_EDIT

ACTION\_DELETE

ACTION\_MAIN

ACTION\_PICK

ACTION\_SEND

Есть еще огромное множество действий, но все они нам не нужны.

Фильтры намерений.

Если намерение запрашивает выполнение какого-либо действия с определенным набором данных, то системе нужно уметь выбрать приложение (или компонент) для обслуживания этого запроса. На помощь приходят фильтры намерений, которые используются для регистрации активностей, сервисов и широковещательных приемников в качестве компонентов, способных выполнять заданные действия с конкретным видом данных.

В фильтре намерений декларируется только три составляющих объекта Intent: действие, данные, категория.

Например, в любом приложении есть главная активность, которая устанавливается как точка входа для задания:

<activity

android:name=".HelloWorldActivity"

android:label="@string/app\_name">

<intent-filter>

<action android:name="android.intent.action.MAIN"

<category android:name="android.intent.category.LAUNCHER"

</intent-filter>

</activity>

* action - Использует атрибут android:name, чтобы задать название действия, которое будет обслуживаться. Каждый фильтр намерений должен иметь один (и только один) тег action. Действия должны быть представлены в виде уникальных строк, которые сами себя описывают.
* сategory - Использует атрибут android:name, чтобы указать, при каких обстоятельствах должно обслуживаться действие. Каждый тег intent-filter способен содержать несколько тегов category. Вы можете задать собственные категории или же брать стандартные значения, предоставляемые системой.
* data - Этот тег дает возможность указать тип данных, с которым может взаимодействовать ваш компонент. При необходимости можно задать несколько тегов data.

<http://developer.alexanderklimov.ru/android/theory/intent.php>

# 27. Уведомления.

Ответ:

<http://developer.alexanderklimov.ru/android/toast.php>

<http://developer.alexanderklimov.ru/android/notification.php>

Toast-уведомления: Всплывающее уведомление (Toast Notification) является сообщением, которое появляется на поверхности окна приложения, заполняя необходимое ему количество пространства, требуемого для сообщения. При этом текущая деятельность приложения остаётся работоспособной для пользователя. В течение нескольких секунд сообщение плавно закрывается. Всплывающее уведомление также может быть создано службой, работающей в фоновом режиме. Как правило, всплывающее уведомление используется для показа коротких текстовых сообщений.

Пример:

Для создания всплывающего уведомления необходимо инициализировать объект Toast при помощи метода Toast.makeText(), а затем вызвать метод show() для отображения сообщения на экране:

// Java

public void showToast(View view) {

//создаём и отображаем текстовое уведомление

Toast toast = Toast.makeText(getApplicationContext(),

"Текст уведомления",

Toast.LENGTH\_SHORT);

toast.setGravity(Gravity.CENTER, 0, 0);

toast.show();

}

У метода makeText() есть три параметра:

* Контекст приложения
* Текстовое сообщение
* Продолжительность времени показа уведомления. Можно использовать только две константы

### Константы для указания продолжительности показа сообщения

* LENGTH\_SHORT — (По умолчанию) показывает текстовое уведомление на короткий промежуток времени;
* LENGTH\_LONG — показывает текстовое уведомление в течение длительного периода времени.

private static final int LONG\_DELAY = 3500; // 3.5 seconds

private static final int SHORT\_DELAY = 2000; // 2 seconds

-

Кроме toast-уведомлений , существует также другой тип уведомлений, который выводится за пределами вашего приложения, а именно, в верхней части телефона в системной строке состояния в виде значка с небольшим текстом.

Далее пользователь должен сдвинуть строку состояния экрана, чтобы увидеть расширенную информацию об уведомлении - текст, картинку. Также можно прямо в уведомлении сделать какое-то действие - написать ответ, поставить на паузу музыку и т.п. Для привлечения внимания к уведомлению можно подключить звук и вибрацию.

Чтобы создать уведомление в строке состояния, необходимо использовать два класса:

* Notification — определяем свойства уведомления строки состояния: значок, расширенное сообщение и дополнительные параметры настройки (звук и др.)
* NotificationManager — системный сервис Android, который управляет всеми уведомлениями. Экземпляр NotificationManager создаётся при помощи вызова метода from(), а затем, когда надо показать уведомление пользователю, вызывается метод notify()

Выводится уведомление с помощью метода notify() .

Показываем уведомление с возможностью взаимодействия с ним (пример кода):

// Идентификатор уведомления

private static final int NOTIFY\_ID = 101;

// Идентификатор канала

private static String CHANNEL\_ID = "Cat channel";



## Текущие и настойчивые уведомления

Вы можете делать уведомления текущими и/или настойчивыми, устанавливая флаги FLAG\_INSISTENT и FLAG\_ONGOING\_EVENT. Уведомления, помеченные как текущие, используются для представления событий, которые выполняются в данный момент времени (например, загрузка файла, фоновое проигрывание музыки). Текущие уведомления необходимы для сервисов, работающих на переднем плане. Пример установки флагов:

notification.flags = notification.flags | Notification.FLAG\_ONGOING\_EVENT;

# 28. Адаптация приложения к различным размерам экрана. Использование альтернативных приложений. Квалификаторы.

Ответ:

Android - это не только смартфоны, это и часы, и планшеты и android-auto (автомобили) и все эти устройства имеют очень разные размеры экрана (физические, разрешение экрана и соотношения сторон). Также уже существуют устройства с несколькими экранами (складные, раздвижные). Это всё усложняет разработку, тк приложение должно корректно работать на разных экранах.

Адаптивный layout — это layout, который предназначен для работы с различными размерами и ориентациями экрана, а также с различными устройствами, на различных языках и версиях Android.

**Адаптивный веб-дизайн (Adaptive Web Design, AWD)**

При проектировании сайта в адаптивном дизайне разрабатывается несколько статичных макетов для разных типов устройств. Шаблоны содержат структуру размещения элементов в зависимости от ширины дисплея. Обычно макеты создают для нескольких стандартных величин ширины, например: 320 px, 480 px, 768 px, 960 px, 1200 px, 1600px. Скрипты определяют размеры экрана пользователя и загружают соответствующий шаблон на основе этих контрольных точек.

В адаптивных макетах учитываются особенности различных устройств. Например, для десктопных экранов – только горизонтальная ориентация и большая область просмотра, а для планшетов и смартфонов – сенсорное управление и возможность поворота девайса для просмотра.

Адаптация размеров компонентов в android studio:

Чтобы создать масштабируемый макет, способный адаптироваться к разным экранам, используйте в качестве значений ширины и высоты отдельных компонентов представления параметры **"wrap\_content"и "match\_parent"**. Если используется "wrap\_content", для ширины или высоты представления устанавливается минимальное значение, позволяющее уместить содержание на экран, а параметр "match\_parent"(известный как "fill\_parent" в API до 8 уровня) служит для растягивания компонента по размеру родительского представления. Если указать параметры "wrap\_content" и "match\_parent" вместо строго заданных размеров, в представлениях будет использоваться минимально необходимое место или они будут растягиваться на всю доступную длину и ширину соответственно.

**RelativeLayout.** С помощью вложенных экземпляров объекта LinearLayout и параметров "wrap\_content" и"match\_parent" можно создавать достаточно сложные макеты. Однако LinearLayout не дает возможности точно управлять взаимным расположением дочерних представлений: в LinearLayout они просто помещаются в ряд друг за другом. Если необходимо расположить дочерние представления иным образом, используйте объект **RelativeLayout**, позволяющий задать относительные позиции компонентов. Например, одно дочернее представление можно выровнять по левому краю экрана, а другое – по правому.

**Альтернативные ресурсы**

Большинство приложений предоставляет альтернативные ресурсы для поддержки определённых конфигураций устройств. Например, в вашем приложении должны быть альтернативные drawable-ресурсы для разных разрешений экрана и альтернативные строковые ресурсы для различных языков. Во время запуска приложения Android определит текущую конфигурацию устройства и загрузит соответствующие ресурсы. Если для какой-либо конфигурации устройства нет доступных ресурсов, Android воспользуется ресурсами приложения по умолчанию.

Как и ресурсы по умолчанию, альтернативные ресурсы хранятся в папках внутри res. Для этих папок используется следующий шаблон названия:

resource\_name-config\_qualifier

* resource\_name — это имя папки для этого типа ресурса (см. табл. выше) Например, drawable или values.
* config\_qualifier определяет конфигурацию устройства, для которого будут использоваться ресурсы. Все возможные квалификаторы рассматриваются здесь (таблица 2).
* Чтобы добавить несколько квалификаторов к одному имени папки, разделите их с помощью тире. Если вы используете несколько квалификаторов для папки ресурса, их нужно расположить в порядке их перечисления в таблице.

Примеры с одним квалификатором:

* Строковые ресурсы, локализованные на японский язык, будут находиться в файле strings.xml внутри папки values-ja в папке res (сокращенно res/values-ja/strings.xml). Строковые ресурсы по умолчанию (ресурсы, которые должны использоваться, когда не найдено специфичных для языка ресурсов) будут находиться в файле res/values/strings.xml. Обратите внимание, что XML-файлы имеют одинаковые имена, в данном случае — strings.xml.
* Стили компонентов для API уровня 21 и выше будут находиться в файле res/values-v21/styles.xml. Ресурсы стилей по умолчанию находятся в res/values/styles.xml.

Пример с несколькими квалификаторами:

* Layout-ресурсы, предназначенные для расположения элементов справа налево и использования в «ночном» режиме, будут находиться в папке res/layout-ldrtl-night/.

**Квалификаторы:**

1. Квалификаторы ориентации экрана. Два режима: port (портретный режим/вертикальный) и land (альбомный режим/горизонтальный).
2. Квалификатор версии платформы определяет минимальный уровень API, поддерживаемый устройством. Например, используйте v11 для API уровня 11 (устройства на Android 3.0 или выше).
3. Квалификатор локализации указывает язык и регион (опционально), который используется в приложении. Данный квалификатор представляет собой двухбуквенный ISO 639-1 код языка, за которым (опять же, опционально) следует двухбуквенный код региона (перед ним будет находиться строчная буква r). Вы можете указать только язык, но только регион указать нельзя.

res/values-fr-rFR/strings.xml  
Строки в этом файле используются на устройствах с французским языком во Франции.

res/mipmap-fr-rCA/strings.xml  
Строки из этого файла будут используются на устройствах с французском языке, но уже в Канаде.

res/layout-ja/content\_main.xml  
Этот layout будет использоваться на устройствах с японским языком.

[https://devcolibri.com/unit/ресурсы-для-разных-экранов/#link4](https://devcolibri.com/unit/%D1%80%D0%B5%D1%81%D1%83%D1%80%D1%81%D1%8B-%D0%B4%D0%BB%D1%8F-%D1%80%D0%B0%D0%B7%D0%BD%D1%8B%D1%85-%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%BE%D0%B2/#link4)

https://www.fandroid.info/podderzhka-raznyh-razmerov-ekrana-pri-razrabotke-android-prilozhenij/

# 29. Хранение пользовательских данных в Android приложении. Виды хранилищ.

Ответ:

Хранение данных в приложении Android - неотъемлемая часть

полноценной работы приложения.

Хранилища данных отвечают на вопрос, где хранить информацию и откуда ее брать, чтобы обеспечить функциональность приложения.

Хранение данных в приложении Android можно осуществить, используя два вида хранилищ:

*Локальное(внутреннее) хранилище*; *Внешнее хранилище; Сетевое хранилище (хранение на сервере (в “облаке”))*

*Локальное* хранилище:

В этом типе хранения данных, можно хранить информацию, пользуясь методом хранения данных в:

*Файле общих настроек; Локальной базе данных*.

Файл общих настроек:

Один из самых простых видов хранения данных. Он позволяет хранить данные, доступные внутри приложения. При удалении приложения, данные тоже удаляются.

Чтобы записать данные в файл настроек, необходимо заполнить ветку сохранения данных. Получить ссылку на файл настроек и оттуда получить объект редактора этого файла.

Файл общих настроек своей структурой похож на словарь.

Информация хранится в виде набора пар <ключ><значение>.

Чтобы получить данные из файла настроек, необходимо указать ссылку на файл, получить оттуда значение по ключу.

Локальная БД:

Локальное хранение данных производится на жестком диске устройства. Это имеет свои плюсы, например доступность к данным в любой момент времени, а так же и минусы, например при повреждении жесткого диска потеряются все файлы.

В качестве реализации хранения данных в локальном хранилище можно привести пример, использования БД в андрод приложении, локально.

Это более сложная и многогранная система работы с данными - реализация реляционной базы данных.

Базы данных позволяют структурировано хранить данные приложения, а также получать необходимые данные в информативном объеме, благодаря запросам.

Локальная база данных всегда доступно для обработки данных, файлы сохраненные в приложении будут доступны только ему, а при удалении приложения с устройства данные удалятся с ним.

“Родным” для приложения Android является реляционная база данных на основе SQLite.

При разработке Android приложения и внедрения базы данных необходимо использовать вспомогательный класс Java (в качестве примера можно привести SQLiteOpenHelper. Он использовался в ходе изучения курса)

Внешнее хранилище:

Другим видом хранилищ является - внешнее хранилище.

В отличие от локального хранилища оно не всегда доступно, поскольку пользователь может подключить внешнее хранилище как USB-накопитель и в некоторых случаях удалить его из устройства.

Также Он доступен для чтения всем, поэтому сохраненные здесь файлы могут быть прочитаны вне вашего контроля.

Сетевое хранилище:

Сетевое хранилище данных более функциональное и что самое главное надежное, по сравнению с локальным.

В локальном хранилище все данные хранятся на 1 устройстве. Потенциальная поломка жесткого диска (в локальном хранилище) или внешнего хранилища угрожает лишиться всех данных приложения.

Сетевое хранилище данных позволяет реализовать взаимодействие приложения с базой данных на сервере благодаря запросам: get, post, put, delete.

Пример взаимодействия:

Один из наиболее удобных способов передачи данных с серверным хранилищем реализуется с помощью файлов формата json.

# 30. Общие настройки. Использование их для хранения данных. Методы доступа.

Ответ:

Нередко приложению требуется сохранять небольшие кусочки данных для дальнейшего использования, например, данные о пользователе, настройки конфигурации и т.д. Для этого в Android существует концепция Preferences или настройки. Настройки представляют собой группу пар ключ-значение, которые используются приложением.

В качестве значений могут выступать данные следующих типов: Boolean, Float, Integer, Long, String, набор строк.

Настройки являются общими для всех activity в приложении, но также могут быть и настройки непосредственно для отдельных activity

Настройки хранятся в xml-файлах в незашифрованном виде в локальном хранилище. Они невидимы, поэтому для простого пользователя недоступны.

При работе с настройками следует учитывать следующие моменты. Так как они хранятся в незашифрованном виде, то не рекомендуется сохранять в них чувствительные данные типа пароля или номеров кредитных карт. Кроме того, они представляют данные, ассоциированные с приложением, и через панель управления приложением в Настройках ОС пользователь может удалить эти данные.

Для работы с разделяемыми настройками в классе Activity (точнее в его базовом классе Context) имеется метод **getSharedPreferences()**:

import android.content.SharedPreferences;

SharedPreferences settings = getSharedPreferences("PreferencesName", MODE\_PRIVATE);

Первый параметр метода указывает на название настроек. В данном случае название - "PreferencesName". Если настроек с подобным названием нет, то они создаются при вызове данного метода. Второй параметр указывает на режим доступа. В данном случае режим описан константой MODE\_PRIVATE

Класс android.content.SharedPreferences предоставляет ряд методов для управления настройками:

* contains(String key): возвращает true, если в настройках сохранено значение с ключом key
* getAll(): возвращает все сохраненные в настройках значения
* getInt(String key, int defValue): возвращает значение типа int с ключом key
* getString(String key, String defValue): возвращает строковое значение с ключом key
* getStringSet(String key, Set<String> defValues): возвращает массив строк с ключом key
* edit(): возвращает объект SharedPreferences.Editor, который используется для редактирования настроек

Для управления настройками используется объект класса **SharedPreferences.Editor**, возвращаемый метод edit(). Он определяет следующие методы:

* clear(): удаляет все настройки
* remove(String key): удаляет из настроек значение с ключом key
* putInt(String key, int value): добавляет в настройки значение int с ключом key
* putString(String key, String value): добавляет в настройки строку с ключом key
* putStringSet(String key, Set<String> values): добавляет в настройки строковый массив
* commit(): подтверждает все изменения в настройках
* apply(): также, как и метод commit(), подтверждает все изменения в настройках, однако измененный объект SharedPreferences вначале сохраняется во временной памяти, и лишь затем в результате асинхронной операции записывается на мобильное устройство

# 31. Использование реляционных хранилищ в андроид приложении. Доступ к данным через поставщиков контента.

Ответ:

Реляционные хранилища данных - база данных (sqlite).

SQLite – это база данных SQL с открытым исходным кодом, которая хранит данные в текстовом файле на устройстве. Android поставляется со встроенной реализацией базы данных SQLite.

SQLite поддерживает все функции реляционной базы данных. Чтобы получить доступ к этой базе данных, вам не нужно устанавливать какие-либо соединения для нее.

Библиотека Android содержит абстрактный класс SQLiteOpenHelper, с помощью которого можно создавать, открывать и обновлять базы данных.

Класс SQLiteOpenHelper содержит два обязательных абстрактных метода:

* onCreate() — вызывается при первом создании базы данных
* onUpgrade() — вызывается при модификации базы данных

Кроме пары главных методов, существует ряд вспомогательных, использующихся по мере необходимости:

* onDowngrade(SQLiteDatabase, int, int);
* onOpen(SQLiteDatabase);
* get Readable Database();
* get Writable Database().

Класс «SQLiteDatabase» – основа управления данными интерфейса. Базовый набор класса «SQLiteDatabase» штатно включает в себя методы:

* query();

Метод «query()» в большинстве случаев используется для выполнения операций чтения данных из общей базы

* insert();

Метод имеет перспективу частого применения, необходим при вставке данных в таблицы SQLite.

* delete();

Работает по принципу аналогичному методу «update()», за исключением конечного результата выполнения. При заявленных параметрах местонахождения таблицы базы данных, «delete» удаляет все информационные составляющие.

* update().

Суть работы метода исходит из его названия – обновление существующих и вновь созданных (загруженных) данных. Для корректной работы необходимо задать методу параметры таблицы базы данных, требующей обновления.

—---------------------------------------------------------------------------------------

Контент-провайдер или "Поставщик содержимого" - это оболочка (wrapper), в которую заключены данные. Если ваше приложение использует базу данных SQLite, то только ваше приложение имеет к ней доступ. Но бывают ситуации, когда данные желательно сделать общими. Простой пример - ваши контакты из телефонной книги тоже содержатся в базе данных, но вы хотите иметь доступ к данным, чтобы ваше приложение тоже могло выводить список контактов. Так как вы не имеете доступа к базе данных чужого приложения, был придуман специальный механизм, позволяющий делиться своими данными всем желающим.

В Android существует возможность выражения источников данных (или поставщиков данных) при помощи передачи состояния представления - REST, в виде абстракций, называемых поставщиками содержимого. Базу данных SQLite можно заключить в поставщик содержимого. Чтобы получить данные из поставщика содержимого или сохранить в нём новую информацию, нужно использовать набор REST-подобных идентификаторов URI.

# 32. Основные понятия языка JavaScript. История, особенности, сравнение с другими языками. Система типов.

Ответ:

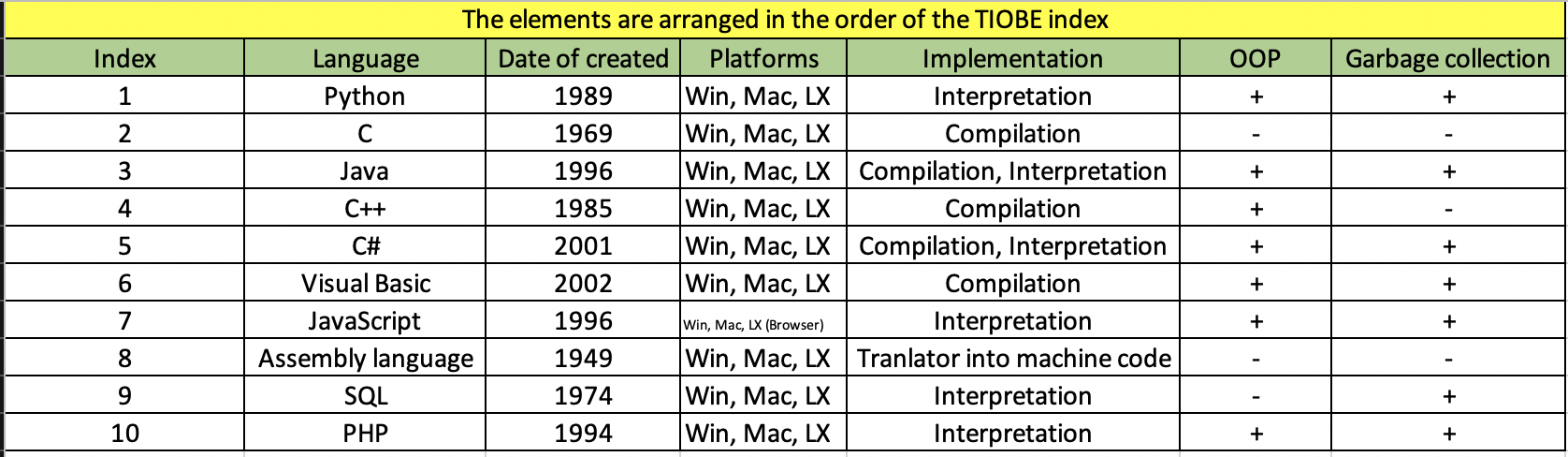
JavaScript был разработан фирмой Netscape в сотрудничестве с Sun Microsystems и анонсирован в 1995 году, сначала назывался LiveScript. JavaScript предназначен для создания интерактивных html-документов. Основные области использования JavaScript:

Создание динамических страниц (содержимое меняется после загрузки);  
Проверка правильности заполнения пользовательских форм.  
Решение "локальных" задач с помощью сценариев.  
JavaScript-код - основа большинства Ajax-приложений.

JavaScript позволяет создавать приложения, выполняемые на стороне клиента, т.е. эти приложения выполняются браузером на компьютере пользователя.

Стандарт языка называется ECMAScript, а сам JS - это его реализация.

Сравнение JS c другими языками (тупизм): (из дз по англу хаха)



Типизация JS: динамическая, слабая, неявная. Типы данных включают в себя:

1. number для любых чисел: целочисленных или чисел с плавающей точкой; целочисленные значения ограничены диапазоном ±(25^3-1).
2. bigint для целых чисел произвольной длины.
3. string для строк. Строка может содержать ноль или больше символов, нет отдельного символьного типа.
4. boolean для true/false.
5. null для неизвестных значений – отдельный тип, имеющий одно значение null.
6. undefined для неприсвоенных значений – отдельный тип, имеющий одно значение undefined.
7. object для более сложных структур данных.
8. symbol для уникальных идентификаторов.

# 33. Основные операторы JavaScript. Условные операторы и циклы.

Ответ: В JavaScript есть следующие типы операторов. Данный подраздел описывает каждый тип и содержит информацию об их приоритетах друг над другом.

* [Операторы присваивания](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B_%D0%BF%D1%80%D0%B8%D1%81%D0%B2%D0%B0%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)
* [Операторы сравнения](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B_%D1%81%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F)
* [Арифметические операторы](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B)
* [Битовые (поразрядные) операторы](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%B1%D0%B8%D1%82%D0%BE%D0%B2%D1%8B%D0%B5_(%D0%BF%D0%BE%D1%80%D0%B0%D0%B7%D1%80%D1%8F%D0%B4%D0%BD%D1%8B%D0%B5)_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B)
* [Логические операторы](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B)
* [Строковые операторы](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%BE%D0%B2%D1%8B%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B)
* [Условный (тернарный) оператор](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D1%83%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D1%8B%D0%B9_(%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%80%D0%BD%D1%8B%D0%B9)_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80)
* [Оператор запятая](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D0%B7%D0%B0%D0%BF%D1%8F%D1%82%D0%B0%D1%8F)
* [Унарные операторы](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D1%83%D0%BD%D0%B0%D1%80%D0%BD%D1%8B%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B)
* [Операторы отношения](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D1%8B_%D0%BE%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D1%8F)
* [Приоритет операторов](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Expressions_and_Operators#%D0%BF%D1%80%D0%B8%D0%BE%D1%80%D0%B8%D1%82%D0%B5%D1%82_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BE%D0%B2)

[Условный оператор](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Conditional_Operator) является единственным оператором JavaScript, который использует три операнда. Оператор принимает одно из двух значений в зависимости от заданного условия. Синтаксис оператора:

*condition* ? *val1* : *val2*

*val1 и val2 обязательно должны что-то возвращать, поэтому в этой конструкции нельзя использовать continue или break*

Если condition (условие) - истина, то оператор принимает значение val1. В противном случае оператор принимает значение val2. Вы можете использовать условный оператор во всех случаях, где может быть использован стандартный оператор.

var status = (age >= 18) ? "adult" : "minor";

Данное выражение присваивает значение "adult" переменной status, если age имеет значение 18 или более. В противном случае переменной status присваивается значение "minor".

Циклы:

for:

Цикл [for](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/for)  повторяет действия, пока не произойдёт какое-либо специальное событие завершения цикла. Оператор for в JavaScript аналогичен оператору for в Java и C. Объявление оператора for выглядит следующим образом:

for ([начало]; [условие]; [шаг]) выражения

for (i = 0; i < a.length; i++) {

if (a[i] == theValue) {

break;

}

}

while:

Цикл [while](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/while) выполняет выражения пока условие истинно. Выглядит он так:

while (условие)

{выражения}

var n = 0;

var x = 0;

while (n < 3) {

n++;

x += n;

}

(почти бесполезный)

do … while:

Цикл [do...while](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/do...while) повторяется пока заданное условие истинно. Оператор do...while имеет вид:

do

выражения

while (условие);

do {

i += 1;

console.log(i);

} while (i < 5);

# 34. Функции в JavaScript. Передача параметров, возврат значений. Стрелочные функции.

Ответ:

Для создания функции используется объявление функции

function имя(параметры) {

...тело...

}

или функциональное выражение

const имя = function(параметры) {

…тело…

}

Для создания стрелочной функции используется конструкция:

const имя = (параметры) => {

…тело…

}

Стрелочные функции:

1. Имеют упрощённый синтаксис

2) Стрелочные функции не имеют собственного контекста выполнения. Они наследуют сущности this и arguments от родительской функции.

3) Стрелочные функции применяются, когда некая функция применяется к каждому элементу списка (с map)

4) Стрелочные функции анонимны

**Параметры**

Параметры представляют собой переменные, с которыми будет работать функция. Параметры задаются на этапе объявления функции и будут содержать передаваемые ей аргументы. Функции могут не иметь параметров. Если при вызове функции параметр не указан, то его значение - undefined. Параметру можно задать значение по умолчанию, указав его после =. Параметры функции указываются в скобках, через запятую. Передаются аргументы в функцию через имяФункции(аргмуент1, аргумент2…).

function showMessage(from, text=’empty’) {

alert(from + ': ' + text);

}

showMessage("Лёша"); // Лёша: empty

Передаваемые функциям аргументы можно представлять в виде массивов. Для того чтобы разобрать эти аргументы можно воспользоваться оператором, который выглядит как три точки («оператор расширения»).

const doSomething = (foo, bar) => {

//сделать что-нибудь

}

const args = [2, 'ho!']

doSomething(...args)

Чтобы избавиться от зависимости последовательности присвоения аргументов от порядка их указания (на случай, если параметров у функции много), используют конструкцию:

const doSomething = ({ foo, bar }) => {

//сделать что-нибудь

console.log(foo) // 2

console.log(bar) // 'ho!'

}

const args = { foo: 2, bar: 'ho!' }

doSomething(args)

**Возврат значения**

Выполнение функции завершается после выполнения содержащегося в ней кода или после ключевого слова return. Вызовов return может быть несколько, например, если в функции есть условное ветвление. Все функции возвращают некое значение. Если команда возврата явно не задана — функция возвратит undefined. Результат-значение функции можно передавать и в другие функции в виде аргумента.

Пример простой функции:

function sum(a, b) {

return a + b;

}

let result = sum(1, 2);

alert( result ); // 3

Из функции можно возвращать лишь одно значение. Для того чтобы получить возможность возврата нескольких значений, возвращать их нужно либо в виде объекта, используя объектный литерал, либо в виде массива.

const doSomething = () => {

return ['Roger', 6]

}

const [ name, age ] = doSomething()

console.log(name, age) //Roger 6

ИЛИ

const doSomething = () => {

return {name: 'Roger', age: 6}

}

const { name, age } = doSomething()

console.log(name, age) //Roger 6

# 35. Объектная модель JavaScript. Прототипы. Классы в ES6. Наследование. Объектные литералы.

Ответ:

В JavaScript определяется функция-конструктор для создания объектов с начальным набором свойств и значений. Для создания объекта используется оператор new.

Поля класса и его методы начинаются с указателя this.

**function** Person(firstName, lastName, dob) {

*// Set object properties*

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.dob = **new** Date(dob); *// Set to actual date object using Date constructor*

**this**.getBirthYear = **function**(){

**return** **this**.dob.getFullYear();

}

**this**.getFullName = **function**() {

**return** `${**this**.firstName} ${**this**.lastName}`

}

}

К любому объекту можно добавить свойство. При этом оно будет только у этого объекта, а не у всех объектов этого класса.

var real\_man = new Person(‘Vladimir’, ‘Putin’)

real\_man.power = ‘infinity’

ES6 добавляет синтаксический сахар в виде объявления класса схожего с другими языками программирования.

class Person {

constructor(firstName, lastName, dob) {

this.firstName = firstName;

this.lastName = lastName;

this.dob = new Date(dob);

}

getBirthYear() {

return this.dob.getFullYear();

}

getFullName() {

return `${this.firstName} ${this.lastName}`

}

}

Наследование происходит с помощью ключевого слова extends

class Rabbit extends Animal {

hide() {

alert(`${this.name} прячется!`);

}

}

Объектный литерал - это неупорядоченная коллекция свойств, имеющих вид ключ: значение. Ключи может быть строка или идентификатор. Ключ обязательно должен быть уникальным. Объектные литералы могут быть вложенными.

var rectangle = {

upperLeft: { x: 2, y: 2 },

lowerRight: { x: 4, y: 4 }

};

# 36. Текстовый ввод-вывод в JavaScript. Использование консоли как отладчика.

Ответ:

Для ввода информации можно использовать элемент Input.

First name: <input id="first\_name">

Last name: <input id="last\_name">

Для вывода информации можно использовать такие элементы как Alert или же Console.

Alert вызывается в коде для отображения любой информации в том числе для отладки, но для каждого алерта нужно подтверждение, что может вызывать трудности в отладке веб-приложения.

Alert('Любой текст или операция в js файле')

Для отладки проще использовать консоль браузера так как в ней можно вывести ту же информацию, что и в Alert, она более компактно хранится и она будет сохранятся на протяжении всей сессии работы со страницей.

Console.log(‘Любой текст или операция в js файле’)

# 37. Массивы и функции высших порядков в Javascript. Деструктуризация массивов и объектов.

Ответ:

**Массивы**

Для хранения упорядоченных коллекций существует особая структура данных, которая называется массив, Array. В массиве могут храниться элементы любого типа.

Объявление:

Существует два варианта синтаксиса для создания пустого массива:

let arr = new Array();

let arr = [];

Элементы массива нумеруются, начиная с нуля.

Мы можем получить элемент, указав его номер в квадратных скобках:

let fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits[0] ); // Яблоко

alert( fruits[1] ); // Апельсин

alert( fruits[2] ); // Слива

Мы можем заменить элемент:

fruits[2] = 'Груша';

добавить новый к существующему массиву:

fruits[3] = 'Лимон'; // теперь ["Яблоко", "Апельсин", "Груша", "Лимон"]

Общее число элементов массива содержится в его свойстве length:

alert( fruits.length ); // 3

Вывести массив целиком можно при помощи alert.

alert( fruits ); // Яблоко, Апельсин, Слива

**Методы, работающие с концом массива:**

**pop**

Удаляет последний элемент из массива и возвращает его: alert( fruits.pop() ); // удаляем "Груша" и выводим его alert( fruits ); // Яблоко, Апельсин

И fruits.pop() и fruits.at(-1) возвращают последний элемент массива, но fruits.pop() также изменяет массив, удаляя его.

**push**

Добавляет элемент в конец массива:

let fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша

**Методы, работающие с началом массива:**

**shift**

Удаляет из массива первый элемент и возвращает его:

let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.shift() ); // удаляем Яблоко и выводим его

alert( fruits ); // Апельсин, Груша

**unshift**

Добавляет элемент в начало массива:

let fruits = ["Апельсин", "Груша"];

fruits.unshift('Яблоко');

alert( fruits ); // Яблоко, Апельсин, Груша

Методы push и unshift могут добавлять сразу несколько элементов:

let fruits = ["Яблоко"];

fruits.push("Апельсин", "Груша");

fruits.unshift("Ананас", "Лимон");

// ["Ананас", "Лимон", "Яблоко", "Апельсин", "Груша"]

alert( fruits );

Цикл for..of не предоставляет доступа к номеру текущего элемента, только к его значению, но в большинстве случаев этого достаточно. А также это короче.

**Многомерные массивы**

Массивы могут содержать элементы, которые тоже являются массивами. Это можно использовать для создания многомерных массивов, например, для хранения матриц:

let matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

alert( matrix[1][1] ); // 5, центральный элемент

**Функции высшего порядка** — это функции, которые работают с другими функциями, либо принимая их в виде параметров, либо возвращая их. Проще говоря, функцией высшего порядка называется такая функция, которая принимает функцию как аргумент или возвращает функцию в виде выходного значения.

Например, встроенные функции JavaScript Array.prototype.map, Array.prototype.filter и Array.prototype.reduce являются функциями высшего порядка.

Метод Array.prototype.map

Метод map() создаёт новый массив, вызывая, для обработки каждого элемента входного массива, коллбэк, переданный ему в виде аргумента. Этот метод берёт каждое возвращённое коллбэком значение и помещает его в выходной массив.

Функция обратного вызова, передаваемая map(), принимает три аргумента: element (элемент), index(индекс) и array (массив). Рассмотрим примеры.

Пример:



Метод Array.prototype.filter

Метод filter() создаёт, на основе массива, новый массив, в которой попадают элементы исходного массива, соответствующие условию, заданному в переданной этому методу функции обратного вызова. Эта функция принимает, как и в случае с методом map(), 3 аргумента: element, index и array.

Рассмотрим пример, построенный по той же схеме, что и при рассмотрении метода map().

Пример:



Метод Array.prototype.reduce

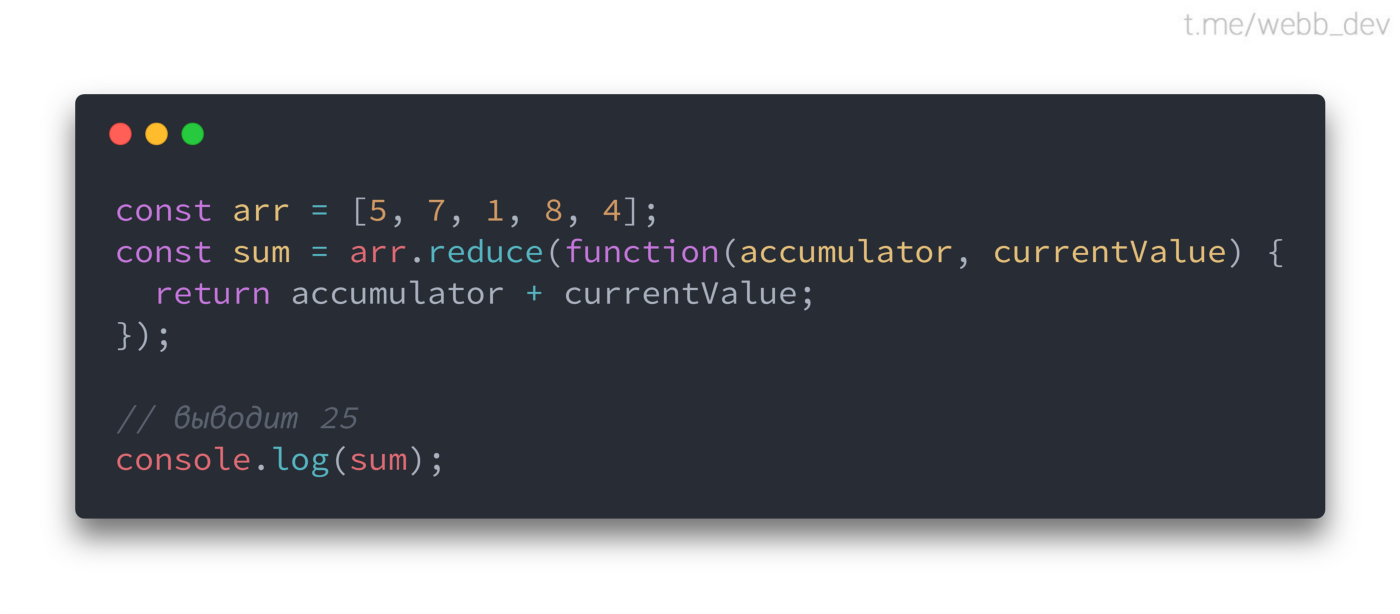
Метод reduce() обрабатывает каждый элемент массива с помощью коллбэка и помещает результат в единственное выходное значение. Этот метод принимает два параметра: коллбэк и необязательное начальное значение (initialValue).

Коллбэк принимает четыре параметра: accumulator (аккумулятор), currentValue (текущее значение), currentIndex (текущий индекс), sourceArray (исходный массив).

Если методу предоставлен параметр initialValue, то, в начале работы метода, accumulator будет равен этому значению, а в currentValue будет записан первый элемент обрабатываемого массива.

Если параметр initialValue методу не предоставлен, то в accumulator будет записан первый элемент массива, а в currentValue — второй.

Пример:



# Деструктуризация массивов

*Деструктуризация* (destructuring assignment) – это особый синтаксис присваивания, при котором можно присвоить массив или объект сразу нескольким переменным, разбив его на части.

Пример деструктуризации массива:

'use strict';

let [firstName, lastName] = ["Илья", "Кантор"];

alert(firstName); // Илья

alert(lastName); // Кантор

При таком присвоении первое значение массива пойдёт в переменную firstName, второе – в lastName, а последующие (если есть) – будут отброшены.

Ненужные элементы массива также можно отбросить, поставив лишнюю запятую.

# Деструктуризация объектов

Деструктуризацию можно использовать и с объектами. При этом мы указываем, какие свойства в какие переменные должны «идти».

Базовый синтаксис:

let {var1, var2} = {var1: …, var2: …};

Объект справа – уже существующий, который мы хотим разбить на переменные. А слева – список переменных, в которые нужно соответствующие свойства записать.

Например:

'use strict';

let options = {

title: "Меню",

width: 100,

height: 200

};

let {title, width, height} = options;

alert(title); // Меню

alert(width); // 100

alert(height); // 200

Как видно, свойства options.title, options.width и options.height автоматически присвоились соответствующим переменным.

# 38. Объектная модель документа. Основные приемы работы.

Ответ:

**DOM (Document Object Model/Объектная модель документа)** – модель документа как объекта, создаваемая веб-браузером в памяти вашего устройства на основании того HTML-кода, который был получен от сервера.

В ближайшем приближении DOM – это "дерево узлов" (node tree). У него единый корень, который разветвляется на множество дочерних ветвей, каждая из которых может ветвиться сама и заканчивается "листьями". Корень – это элемент html, а ветви – вложенные элементы.

Другими словами, непосредственно сам документ в виде HTML браузер не показывают в исходном виде. Сначала идет запрос к серверу, а полученный ответ разбирается для построения дерева страницы или DOM-дерева.

Схема такова:

1. Сервер создает HTML-код страницы или отдает его (если это простой статический сайт);
2. Браузер получает код и разбирает на элементы дерева;
3. При необходимости подключается JavaScript, если он используется, чтобы изменить поведение тегов и их содержимого в зависимости от воздействий пользователя;
4. DOM-дерево отображается во вкладке браузера в том виде, который задуман разработчиками.

**Основные понятия модели DOM**

1. **Дерево** (веб-страница может быть представлена как иерархическое перевернутое дерево, начинающееся с главного элемента и расширяющееся к низу. У каждого объекта дерева есть родитель, который может быть пустым, т.е. null, и дети, если он находится не в самом низу);
2. **Наследник** (любой дочерний элемент узла дерева. Наследник может быть инклюзивным, т.е. не прямым. Пример: у тега <html> есть дочерний элемент <body>, внутри которого содержится элемент <article>, тогда тег <article> это не прямой наследник <html>);
3. **Предок** (если у объекта DOM-дерева есть потомки, то для них он является предком. Здесь также возможна инклюзивность. Из примера выше у тега <article> есть прямой предок – тег <body>, а также инклюзивный родитель – тег <html>);
4. **Братья / сестры** (в стандарте используется термин sibling, что с английского переводится как родной брат или сестра. Так объект подразумевает наличие у него «соседей», стоящих на там же уровне дерева и единого предка, не являющегося пустым);
5. **События** (к объектам дерева можно применять разные события при помощи JavaScript. Они могут срабатывать на действия пользователя или сетевую активность. Так, при наведении мыши, клике, нажатии клавиши или обрыве связи возможно разное поведение всего сайта или его элементов);
6. **Узел** (все элементы DOM-дерева являются узлами, нодами, nodes. Их можно создавать, изменять, вызывать свойства и методы с помощью JS).

Для **навигации по DOM-дереву** следует иметь представление о следующих понятиях:

1. **Корневые элементы**:

а) document.documentElement (самый верхний узел, соответствует тегу <html>, включает в себя всё содержимое документа);

б) document.head (заголовочная часть web-страницы);

в) document.body (тело документа, содержимое тега <body>).

2. **Узел-родитель** (parentNode – непосредственный потомок конкретного объекта)

3. **Дети, потомки**:

а) childNodes (коллекция потомков определенного объекта);

б) firstChild (первый ребенок);

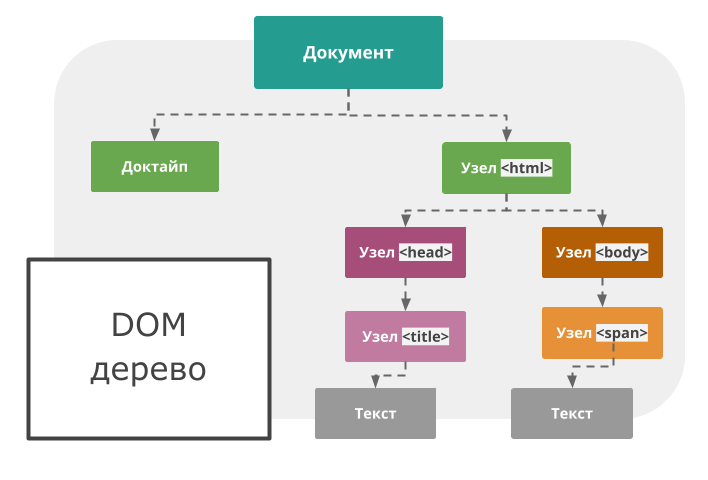
в) lastChild (последний дочерний элемент).

4. **Соседи** (один уровень иерархии):

а) nextSibling (следующий сосед того же родителя);

б) previousSibling (предыдущий элемент внутри родителя, находящийся на том же уровне, что и изначальный).

Структура DOM:

При помощи JavaScript мы можем не только находить элементы в web-документе или менять их свойства, но и вносить изменения в структуру DOM-дерева:

1. Создавать новые теги
2. Удалять имеющиеся
3. Заменять их на другие

Для этого применяются следующие инструменты:



# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 39. Событийная модель в JavaScript.

Ответ:

Браузер постоянно сообщает о разного рода событиях (events), и к любому из них есть способ определить некоторый обработчик. Он представляет собой функцию, которая реагирует на ивент. Приведем частичный перечень событий:



Саму обработку событий можно разделить на: обработку событий через свойство тегов в HTML коде, а также с помощью специальных методов  JS.

Обработка событий с помощью JS:

Сюда относятся следующие методы:

* addEventListener()
* removeEventListener()

Функция addEventListener() позволяет присваивать элементам более одного обработчика событий, а также работает не только с тегами, но и любыми DOM-объектами.

Параметр event функции-обработчика:

Зачастую требуется не просто обработать некое событие, но и понять детали того, что именно произошло. В этом случае в функции-обработчики передается параметр event.

Этапы события (погружение, достижение цели, всплытие):

Когда на странице расположено большое количество объектов, то нажатие на одном из них (если у него есть ряд предков) вызовет срабатывание обработчиков (если они присутствуют) у его родителей. Так, если сделан клик по абзацу внутри тега <article>, то и он это заметит. Данный принцип называется всплытием.

С другой стороны, мы не всегда приветствуем такое поведение. Может потребоваться реакция браузера только на конкретное событие, а не на всю цепочку родителей. Чтобы узнать, какой элемент вызвал изначальное срабатывание события, используют свойство event.target. Даже если ивент присвоен верхнему элементу, то выявится конкретный дочерний объект, который был задействован.

# 40. Стек технологий веб-разработки. Основные языки разработки. Клиентская и серверная разработка.

Ответ:

Стек технологий — это набор технологий, на основе которых разрабатывается сайт или приложение. Стек включает в себя языки программирования, фреймворки (программная среда для разработки), системы управления базами данных, компиляторы и т.д, то есть клиентскую (со стороны пользователя) и серверную (логика приложения) части. Выбор конкретного стека зависит от архитектуры проекта, сложности и функциональности сайта, системных требований.

Клиентская разработка (фронтэнд) представляет собой интерактивную часть программы, которая исполняется в веб-браузере на компьютере, смартфоне или планшете. Клиентская часть реализует пользовательский интерфейс веб-приложения и загружается в виде динамических веб-страниц. Есть 3 главных элемента разработки клиентской части любого веб-приложения:

* JavaScript — язык программирования, который отвечает за интерактивную часть веб-приложения.
* HTML — язык разметки документов, который необходим для правильного отображения веб-приложения в браузере.
* CSS — формальный язык, который необходим для правильной стилизации веб-приложения.

Если говорить о фреймворках, чаще всего для разработки клиентской части любого веб-приложения используются Bootstrap и React.js.

Под серверной частью понимают набор аппаратно-программных средств, с помощью которых реализована логика работы приложения. Это то, что происходит вне браузера и компьютера пользователя. К бэкенду относится панель администрирования, управление данными, логика их передачи по запросам фронтенда. Для разработки серверной части используются:

* Язык бэкэнд-программирования.
* Фреймворки.
* База данных.
* Сервер. Также возможна разработка веб-приложения с бессерверной архитектурой.

Основные языки разработки

Для фронтенда:

* HTML - язык разметки веб-страниц. С его помощью создают абзацы, заголовки и гиперссылки, добавляют изображения и другие объекты. HTML-код “объясняет” браузеру, что отобразить на странице. У каждого элемента есть свой тег, заключённый в скобки.
* CSS - язык разметки стилей. Позволяет применять различные стили и дизайн к странице и ее компонентам.
* JavaScript/JSX - язык программирования, который помогает реализовывать сложное поведение веб-страницы. В большинстве случаев JS используется для создания адаптивных интерактивных элементов для веб-страниц, которые улучшают взаимодействие с пользователем. С помощью JavaScript можно создавать меню, анимацию, видеоплееры, интерактивные карты и даже простые игры в браузере.
* React JS/ Angular / Vue - фреймворки для удобства работы с кодом. React — библиотека с открытым кодом для создания пользовательских интерфейсов. С помощью React разработчики создают веб-приложения, которые изменяют отображение без перезагрузки страницы. Благодаря этому приложения быстро реагируют на действия пользователя, например, когда они заполняют формы, применяют фильтры, добавляют товары в корзину и так далее. Angular от Google предоставляет разработчикам лучшие условия для объединения JavaScript с HTML и CSS. К веб приложениям на Angular.js относятся Google и Youtube. Vue JS подходит для постепенного внедрения, он легко интегрируется с другими библиотеками и существующими проектами.

Для бэкэнда:

* node js - кроссплатформенная среда, которая выполняет код JavaScript вне браузера. Node.js позволяет разработчикам использовать JavaScript, чтобы получить инструменты командной строки.
* php - принимает входящий запрос от веб-сервера, выполняет сценарий и возвращает результат в виде готового HTML-кода. Сервер отправляет этот результат в браузер пользователю, который, в свою очередь, отображает её пользователю.
* python - предлагает большой спектр библиотечных инструментов и пакетов услуг, что способствует уменьшению времени на разработку приложений. Фреймворки python, например, django, используются для серверных технологий, помогая в маршрутизации URL-адресов, HTTP-запросах, доступе к базам данных и веб-безопасности.
* java - ​​универсальный язык программирования, который часто используется для веб-разработки, а также для разработки под Android. Среди преимуществ веб- приложений на Java — кроссплатформенность, мультифункциональность, надёжность и гибкость.

# 41. Стилизация веб-страницы под ОС Android. Адаптивная верстка. Инструменты адаптивной верстки. Подход “mobile first”.

Ответ:

Стилизация веб-страницы под ОС Android:

Чтобы веб-страница была адаптирована под ОС Android нужно указывать ширину, высоту и расстояние между блоками в процентах.

Если выполнить это условие, то при открытии сайта в ОС Android все блоки и текст будут уменьшены по сравнению, если открыть тот же веб-сайт на ноутбуке.

Адаптивная верстка — это такой способ создания страниц, при котором они автоматически подстраиваются под размеры и ориентацию экрана устройства, а их дизайн варьируется в зависимости от действий пользователя.

Инструменты адаптивной верстки:

* Инструментарий Bootstrap

Эта платформа — своеобразный скелет для адаптивных веб-сайтов. Нужно просто поэтапно сформировать структуру HTML и правильно классифицировать созданные объекты. Далее фронтенд отобразит готовые блоки с оформлением.

* Grid CSS

Grid модуль для CSS был разработан рабочей группой CSS для того, чтобы сделать создание шаблонов в CSS максимально удобным.

* Flexbox

Flexbox— это современный нативный метод верстки страниц сайта с помощью каскадных таблиц стилей.

* Startup

Это вспомогательная программа-проектировщик, которую используют в помощь Bootstrap, чтобы создавать темы из готовых блоков. В результате будет получен сайт с HTML, CSS и JS файлами. Итоговый макет будет предусматривать адаптацию под разные расширения средств просмотра.

* Сервис Responsinator

С помощью этого сервиса можно увидеть, как будет отображаться сайт на разных мониторах. Для этого достаточно ввести URL сайта в верхней панели сервиса.

* Инструмент Caniuse

Сервис, позволяющий свести к минимуму ошибки при адаптивной верстке. В Caniuse можно найти информацию о способности браузера поддерживать те или иные свойства сайта: например, CSS свойство sticky или формат шрифта ttf.

Термин “mobile first” последнее время довольно популярен. Он означает, что нужно начать с создания макета для мобильного телефона, но оптимизированного сразу и под экраны с большим разрешением. Таким образом, ваш макет для мобильных телефонов становится главным и основным, и это позволяет не делать других отдельных макетов.

# 42. Медиа запросы CSS. Их использование для верстки страниц для разных типов экранов.

Ответ:

Медиавыражения (медиазапросы) CSS используются для применения разных CSS-стилей для разных устройств по типу отображения (напр. для принтера, монитора или смартфона), для конкретных характеристик устройства (напр. ширины экрана). Медиавыражения являются важным инструментом при создании веб-сайтов с адаптивной версткой, реализующих подход “mobile first”.

Все запросы начинаются с правила @media, после чего следует условие, в котором используются типы носителей, логические операторы и медиа-функции.

Для использования медиазапросов для разных типов экранов после @media прописывается ключевое слово screen, а затем с помощью логических операторов (and, or, not) указывается параметр вёрстки, влияющий на отображение. Для экранов это, чаще всего, width, height, min-width, min-height, max-width и max-height и landscape.

Пример:

/\* Красный фон применится, если ширина области видимости от 320 до 600 пикселей \*/

**body** {

**background-color**: #FFD23F; /\* Желтый фон \*/

}

**@media** screen **and** (**min-width:** 320px) **and** (**max-width:** 600px) {

**body** {

**background-color**: #EE4266; /\* Красный фон \*/

}

}

# 43. CSS Flexbox. Особенности, назначение, основные понятия и приемы использования.

Ответ:

Это новая технология, которая уже имеет достаточно широкую поддержку браузеров. Flexbox предоставляет инструменты для быстрого создания сложных, гибких макетов, и функции, которые были сложны в традиционных методах CSS.

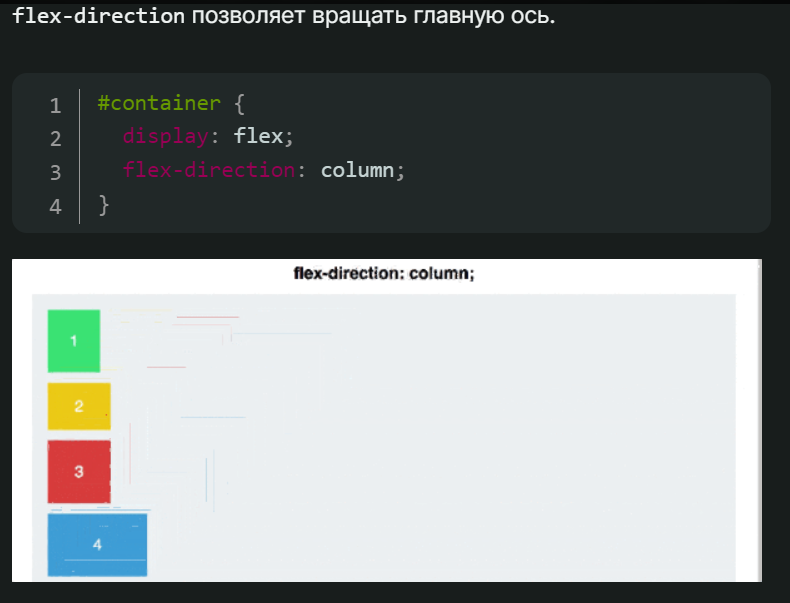
Долгое время единственными надёжными инструментами CSS вёрстки были такие способы как Float (обтекание) и позиционирование.

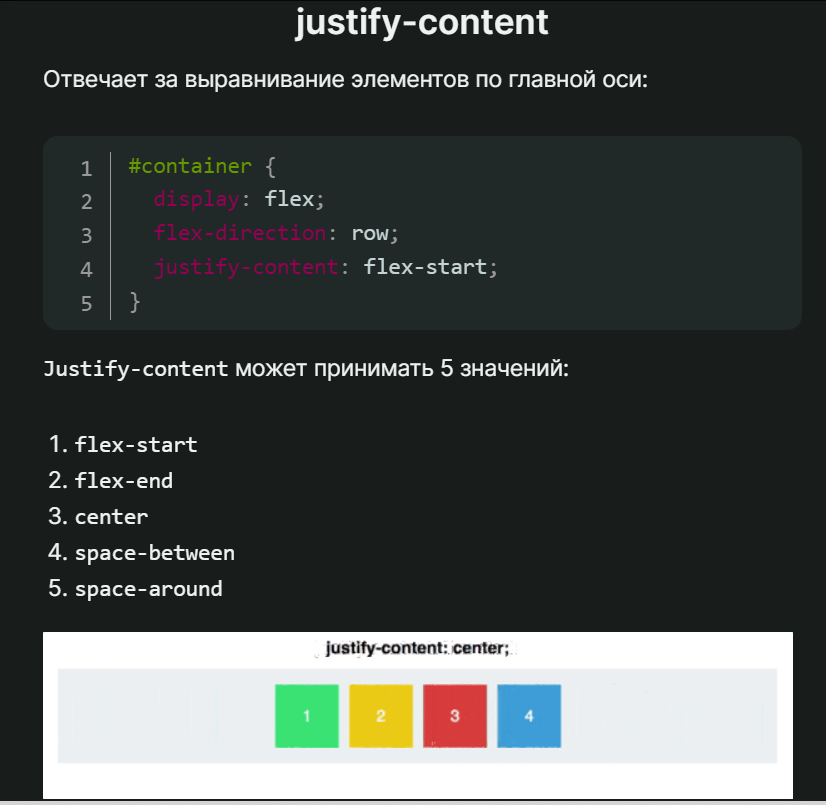
С их помощью сложно или невозможно достичь следующих простых требований к макету:

* Вертикального выравнивания блока внутри родителя.
* Оформления всех детей контейнера так, чтобы они распределили между собой доступную ширину/высоту, независимо от того, сколько ширины/высоты доступно.
* Сделать все колонки в макете одинаковой высоты, даже если наполнение в них различно.

# Примеры (их до пизды на самом деле, вот ссылка: <https://tproger.ru/translations/how-css-flexbox-works/>)

# 





# 44. CSS Grid. Особенности, назначение, основные понятия и приемы использования.

Ответ:

**CSS Grid** – способ гибкого расположения элементов на странице: позволяет создавать структуры, необходимые для обеспечения отзывчивости сайтов на различных устройствах. Элементы Грида (grid items) располагаются вдоль главной (main axis, столбцы) и поперечной (cross axis, строки) осей (т.е. двумерно, как в таблице). Свойства в грид могут быть родительские (у грид-контейнеров, объявленных через display: grid) и дочерние (у грид-элементов).

**Родительские свойства в грид:**

**1)** **grid-template-columns -** используется для определения **количества и ширины** колонок (грид-треков). Можно определять свойства для каждой колонки в отдельности или устанавливать ширину всех колонок с помощью функции repeat(). Размер грид-трека может быть задан любой единицей длины. В грид также существует тип fr (fraction) – доля доступного пространства в грид-контейнере (это гибкая размерность).

**Примечание:**

* Грид может быть явным и неявным. Явный грид – заданный через grid-template-columns и grid-template-rows. Если же размещать что-нибудь вне рамок определённого грида или из-за количества контента требуется большее количество грид-треков, грид создаёт строки и колонки в виде неявного грида. Размер этих треков по умолчанию задаётся автоматически, в зависимости от находящегося в них контента. Но можно и задать размер таких треков через grid-auto-rows и grid-auto-columns.
* При определении размеров колонок или строк может понадобиться задать трекам минимальный или максимальный размер, но при этом быть уверенными, что они треки растянутся, чтобы вместить весь добавленный в них контент. В гриде есть решение этой задачи – функция minmax().
* auto в определении свойства размера означает, что высота/ширина строки растягивается до размера ячейки с самым большим элементом контента.

**2)** **grid-template-rows –** аналогично grid-template-columns, но со строками.

**3)** **column-gap -** используется для добавления отступа между **колонками**.

**4)** **row-gap -** используется для добавления отступов между **строками**.

**5)** **justify-items -** используется для позиционирования грид-элементов внутри грид-контейнера вдоль **главной оси (горизонтально)**. Принимает **4** возможных значения: start, end, center, stretch

**6)** **align-items –** аналогично justify-items, но элементы располагаются вдоль поперечной оси (вертикально).

**7)** **justify-content** - используется для позиционирования самого грида (самой таблицы) внутри грид-контейнера вдоль **основной оси**. Оно принимает **7** возможных значений: start, end, center, stretch, space-between (вплотную к границам), space-around (расстояние между элементом и границей – ½ расстояния между элементами), space-evenly (одинаковое расстояние между элементами и границами)

**8)** **align-content –** аналогично justify-content, но для поперечной оси.

**Дочерние свойства в грид:**

**Примечание:**

При определении грида, на макете появляются линии (грид-линии), с помощью которых можно ориентироваться для работы с макетом. Для определения строки и колонки при их используется два вида единиц: целые числа или ключевое слово span.

**1)** **grid-column: start/end (span) -** позволяет объединять **колонки**.

.container {

**display**: grid;

**gap**: 20px;

**height**: 100vh;

**grid-template-columns**: repeat(12, 1fr); // 12 столбцов, расстояние 1fr

**grid-template-rows**: repeat(12, 1fr);

}

.box-1 {

**grid-column**: span 9; // Элемент будет занимать 9 столбцов из 12 через span

}

.box-2 {

**grid-column**: 1/10; // Элемент будет занимать 9 столбцов из 12 через целочисленные значения

}

**2)** **grid-row: start/end (span) –** свойство, аналогичное grid-column: start/end. Позволяет объединять **строки**.

**3)** **justify-self -** используется для позиционирования **отдельного** грид-элемента (т.е. в пределах грид-ячейки) вдоль **основной оси (горизонтально).** Оно принимает **4** возможных значения: start, end, center, stretch

**4)** **align-self** – свойство, аналогичное justify-self, но вдоль попереченой оси (вертикально).

**Особенностью CSS Grid также являются сокращения для свойств:**

* place-content (сокращение для align-content и justify-content). Пример:

**place-content**: **center** / **end**; // Первое – для align, второе – для justify

* place-items (сокращение для align-items и justify-items). Пример:

**place-items**: **end** / **center**; // Первое – для align, второе – для justify

* place-self (сокращение для align-self и justify-self). Пример:

**place-self**: **start** / **end**; // Первое – для align, второе – для justify

* grid-template (сокращение для grid-template-rows и grid-template-columns). Пример:

**grid-template**: 100**px** 100**px** / 200**px** 200**px**; // Первое – строки, второе - столбцы

* gap / grid-gap (сокращение для row-gap и column-gap). Пример:

**gap**: 20**px** 30**px**;

# 45. Bootstrap 4. Особенности, назначение, основные понятия и приемы использования.

Ответ:

Bootstrap это фреймворк для быстрой верстки адаптивных веб-приложений.

Он построен с использованием Flexbox. Bootstrap исповедует принцип “mobile first”.

Главной особенностью является система сеток. Экран делится на 12 столбцов, что помогает для организации адаптивной вёрстки. Ширину элемента можно задать в несколько колонок. Реализуются четыре класса для устройств с разной шириной:

xs - для смартфонов

sm - для планшетов

md - для десктопов

lg - для широкоформатных устройств.

Для указания адаптивного размера используются классификаторы, аналогичные классам колонок.

Для организации вёрстки используются контейнеры. Они могут быть как фиксированной ширины, так и на всю доступную ширину.

В остальном это просто набор CSS и JavaScript файлов, которые реализуют готовые классы и компоненты.

Bootstrap подходит для быстрой разработки стандартного приложения или прототипирования. Система сеток активно используется разработчиками, однако компоненты и шаблоны Bootstrap занимают много места, что увеличивает вес сборки.

# 46. Использование асинхронных запросов в веб-приложении.

Ответ:

Синхронный JavaScript запрос - это запрос, результат которого возвращается, как только браузер может его обработать.

const btn = document.querySelector('button');

btn.addEventListener('click', () => {

alert('You clicked me!');

let pElem = document.createElement('p');

pElem.textContent = 'This is a newly-added paragraph.';

document.body.appendChild(pElem);

});

Поэтому чаще всего приходится использовать асинхронный JS для работы с веб-приложением. Так, например, когда вы получаете картинку с сервера, мы не можем мгновенно вернуть результат потому что мы не знаем сколько времени займет получение картинки с сервера:

let response = fetch('myImage.png');

let blob = response.blob();

Есть 2 варианта работы с асинхронными запросами - callbacks и promises.

Асинхронные колбэки — это функции, которые определяются как аргументы при вызове функции, которая начнет выполнение кода на заднем фоне. Когда код на заднем фоне завершает свою работу, он вызывает колбэк-функцию, оповещающую, что работа сделана, либо оповещающую о трудностях в завершении работы.

function loadAsset(url, type, callback) {

let xhr = new XMLHttpRequest();

xhr.open('GET', url);

xhr.responseType = type;

xhr.onload = function() {

callback(xhr.response);

};

xhr.send();

}

function displayImage(blob) {

let objectURL = URL.createObjectURL(blob);

let image = document.createElement('img');

image.src = objectURL;

document.body.appendChild(image);

}

loadAsset('coffee.jpg', 'blob', displayImage);

Мы создали функцию displayImage(), которая представляет blob, переданный в неё, как объект URL, и создаёт картинку, в которой отображается URL, добавляя её в элемент документа <body>. Однако, далее мы создаём функцию loadAsset(), которая принимает колбэк-функцию в качестве параметра, вместе с URL для получения данных и типом контента. Для получения данных из URL используется XMLHttpRequest (часто сокращается до аббревиатуры "XHR") , перед тем как передать ответ в колбэк-функцию для дальнейшей обработки. В этом случае колбэк-функция ждёт, пока XHR закончит загрузку данных (используя обработчик события onload) перед отправкой данных в колбэк-функцию.

Промисы — новый стиль написания асинхронного кода, который используется в современных Web API. Хорошим примером является fetch() API, который современнее и эффективнее чем XMLHttpRequest.

fetch('products.json').then(function(response) {

return response.json();

}).then(function(json) {

products = json;

initialize();

}).catch(function(err) {

console.log('Ошибка загрузки: ' + err.message);

});

В примере видно, как fetch() принимает один параметр — URL ресурса, который нужно получить из сети, — и возвращает промис. Промис — это объект, представляющий асинхронную операцию, выполненную удачно или неудачно. Он представляет собой как бы промежуточное состояние.

<https://developer.mozilla.org/ru/docs/Learn/JavaScript/Asynchronous/Introducing>

# 47. Управление загрузкой содержимого с помощью Ajax. Объект XMLHttpRequest. Fetch API

Ответ:

# Управление загрузкой содержимого с помощью Ajax

AJAX (англ. *Asynchronous JavaScript and XML* — асинхронный JavaScript и XML) — это не технология сама по себе, а термин, который описывает «новый» подход к использованию существующих технологий вместе.

При использовании AJAX нет необходимости обновлять каждый раз всю страницу, так как обновляется только ее конкретная часть.

***Загрузка с помощью Ajax***

### **jQuery(..).load**

Начнем с самого простого — загрузка HTML кода в необходимый нам DOM элемент на странице. Для этой цели нам подойдет метод load. Данный метод может принимать следующие параметры:

1. url запрашиваемой страницы
2. передаваемые данные (необязательный параметр)
3. функция которой будет скормлен результат (необязательный параметр)

Приведу пример JavaScript кода:

// по окончанию загрузки страницы

$(document).ready(function(){

// вешаем на клик по элементу с id = example-1

$('#example-1').click(function(){

// загрузку HTML кода из файла example.html

$(this).load('ajax/example.html');

})

});

Пример подгружаемых данных (содержимое файла *example.html*):

Example<[br](http://december.com/html/4/element/br.html)/>

Data Loaded By AJAX<[br](http://december.com/html/4/element/br.html)/>

Bye-Bye

### **jQuery.ajax**

Это самый основной метод, а все последующие методы лишь обертки для метода jQuery.ajax. У данного метода лишь один входной параметр — объект включающий в себя все настройки (выделены параметры которые стоит запомнить):

* async — асинхронность запроса, по умолчанию true
* cache — вкл/выкл кэширование данных браузером, по умолчанию true
* contentType — по умолчанию «application/x-www-form-urlencoded»
* data — передаваемые данные — строка иль объект
* dataFilter — фильтр для входных данных
* dataType — тип данных возвращаемых в callback функцию (xml, html, script, json, text, \_default)
* global — тригер — отвечает за использование глобальных AJAX Event'ов, по умолчанию true
* ifModified — тригер — проверяет были ли изменения в ответе сервера, дабы не слать еще запрос, по умолчанию false
* jsonp — переустановить имя callback функции для работы с JSONP (по умолчанию генерируется на лету)
* processData — по умолчанию отправляемые данный заворачиваются в объект, и отправляются как «application/x-www-form-urlencoded», если надо иначе — отключаем
* scriptCharset — кодировочка — актуально для JSONP и подгрузки JavaScript'ов
* timeout — время таймаут в миллисекундах
* type — GET либо POST
* url — url запрашиваемой страницы

### **jQuery.get**

Загружает страницу, используя для передачи данных GET запрос. Может принимать следующие параметры:

1. url запрашиваемой страницы
2. передаваемые данные (необязательный параметр)
3. callback функция, которой будет скормлен результат (необязательный параметр)
4. тип данных возвращаемых в callback функцию (xml, html, script, json, text, \_default)

**jQuery.getJSON**

Загружает данные в формате JSON (удобней и быстрее нежели XML). Может принимать следующие параметры:

1. url запрашиваемой страницы
2. передаваемые данные (необязательный параметр)
3. callback функция, которой будет скормлен результат (необязательный параметр)

JavaScript:

$(document).ready(function(){ // по завершению загрузки страницы

$('#example-4').click(function(){ // вешаем на клик по элементу с id = example-4

$.getJSON('ajax/example.json', {}, function(json){ // загрузку JSON данных из файла example.json

$('#example-4').html('');

// заполняем DOM элемент данными из JSON объекта

$('#example-4').append('To: ' + json.note.to + '<br/>')

.append('From: ' + json.note.from + '<br/>')

.append('<b>' + json.note.heading + '</b><br/>')

.append( json.note.body + '<br/>');

});

})

});

Файл *example.json*:

{

note:{

to:'Tove',

from:'Jani',

heading:'Reminder',

body:'Don\'t forget me this weekend!'

}

}

### **Отправка Формы**

Для отправки формы посредством jQuery можно использовать любой из перечисленных способов, а вот для удобства «сбора» данных из формы лучше использовать плагин jQuery Form

### **Отправка Файлов**

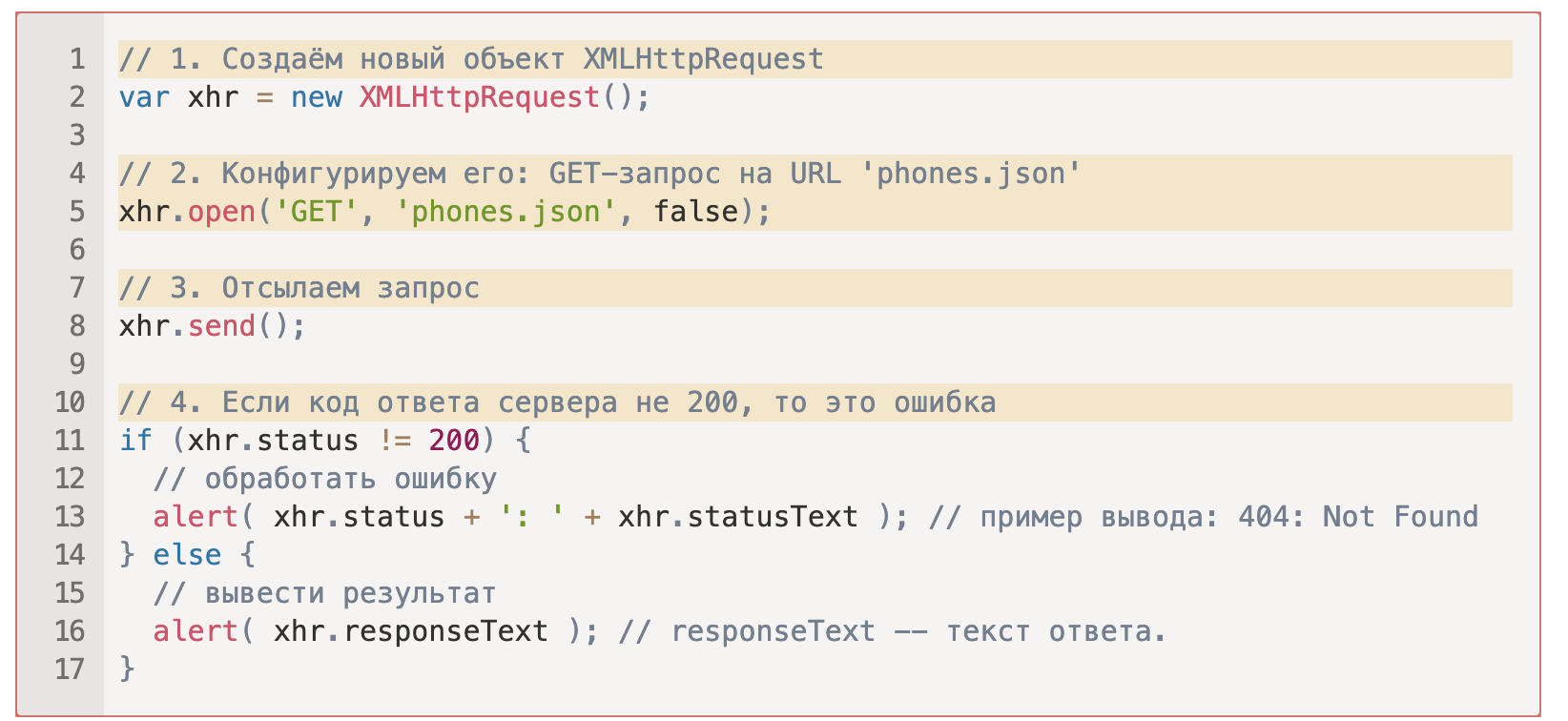
Для отправки файлов посредством jQuery можно использовать плагин Ajax File Upload или One Click Upload

**Объект XMLHttpRequest**

Объект XMLHttpRequest (или, как его кратко называют, «XHR») даёт возможность из JavaScript делать HTTP-запросы к серверу без перезагрузки страницы.

Несмотря на слово «XML» в названии, XMLHttpRequest может работать с любыми данными, а не только с XML.

Как правило, XMLHttpRequest используют для загрузки данных.

Для начала посмотрим на пример использования, который загружает файл phones.json из текущей директории и выдаёт его содержимое:**Fetch API**

Fetch API предоставляет интерфейс для получения ресурсов (в том числе по сети)

Fetch обеспечивает обобщенное определение объектов Request и Response (и других вещей, связанных с сетевыми запросами). Это позволит им использоваться везде, где необходимо в будущем, в том числе и для service workers, Cache API и других подобных технологий, которые обрабатывают или изменяют запросы (requests) и ответы (responses), а так же в любых других случаях, которые могут потребовать от вас генерировать свой собственный ответ программно.

Это также предоставляет определение в отношение таких понятий, как CORS и семантика заголовков HTTP origin, заменяя их обособленные определения где бы то ни было.

Чтобы создать запрос и получить данные, используется метод GlobalFetch.fetch . Он реализован во множестве интерфейсов, в том числе в Window и WorkerGlobalScope . Это позволяет использовать его практически в любом контексте для получения данных.

Метод fetch() принимает один обязательный аргумент — путь к данным, которые вы хотите получить. Он возвращает promise, который разрешается в (Response) независимо от того, был ли запрос удачным. Вы можете также передать во втором аргументе необязательный объект с указанием опций (см. Request.)

Как только Response выполнится успешно, становятся доступными несколько методов для определения тела контента и, как его содержимое должно быть обработано (см. Body.)

Вы можете создавать запрос и ответ непосредственно, используя конструкторы Request() и Response(), но маловероятно, что в этом есть необходимость. Напротив, более вероятно, что они будут созданы как результат работы другого API

# 48. Базовые принципы работы веб-приложения в оффлайн режиме.

Ответ:

Всё чаще возникает задача научить frontend-приложение работать в автономном режиме. Это значит придать web-приложению свойство mobile- или desktop-программы — функционировать в отсутствии связи с Интернет, а также в случае отказа сервера.

Цель — оградить пользователя от проблем соединения на его устройстве.

Для этого нужно инициализировать приложение в браузере и затем закэшировать ресурсы, минимально необходимые для функционирования. После этого приложение возможно запустить из кэша в случае недоступности сервера.

Решение задачи заключается в следующем:

1. при первом посещении web-страницы, получить с сервера “статические” ресурсы в виде html-, css-, js-файлов, спрайтов и пр.;
2. закэшировать ресурсы на стороне клиента средствами браузера;
3. в дальнейшем при запросе этих же файлов выдавать их из кэша в том случае, если отсутствует соединение с сервером;
4. обновлять изменённые ресурсы в кэше.

**ServiceWorker**

ServiceWorker — это новая технология, позволяющая запускать javascript-код в браузере в фоновом режиме — аналог сервисов (служб) в операционных системах. ServiceWorker запускается с web-ресурса и продолжает работать в браузере независимо от приложения, которое его инициализировало.

Часто цель применения ServiceWorker это получение push-уведомлений в браузере и контроль кэшируемых ресурсов, последнее как раз наш случай.

**CacheStorage**

CacheStorage представляет собой контейнер для хранения кэша сетевых ресурсов. Глобальный объект CacheStorage доступен по имени caches. Его составляющие это объекты типа Cache.

Cache — это именованное хранилище из пар: объект Request — объект Response. Для каждого закэшированного ресурса экземпляр Cache будет хранить request и response, созданные функцией fetch.

Как это выглядит на практике?

Теперь разберём всё это на небольшом тестовом приложении. Допустим, что файлы расположены на локальном сервере по адресу localhost/test\_serviceworker. Поставим задачи. Для того, чтобы управлять кэшированием, необходимо:

1. создать serviceWorker, который будет работать в браузере, независимо от наличия/отсутствия доступа к сети;
2. сложить в caches ресурсы, которые должны быть закэшированы;
3. в коде serviceWorker’а повесить обработчик на событие fetch — событие, возникающее при запросе сетевого ресурса;
4. построить в обработчике логику выдачи и обновления кэша.

**Выводы**

Существенный плюс — гибкая система кэширования. Что и когда кэшировать — всё в наших руках (попрощайтесь с ApplicationCache!).

Не стоит использовать такую методику для “нестатических” ресурсов, т.е. для данных получаемых от сервера. Лучше это делать не в сервисе, а внутри front-приложения на уровне модели и использовать при этом, например, localStorage.

Существует такой нюанс: serviceWorker разрешено загрузить только по HTTPS либо с локального сервера. Неприятное ограничение, но в то же время это правильно. HTTPS становится стандартом для популярных сервисов.

Самый главный недостаток. ServiceWorker и CacheStorage — обе технологии экспериментальные на текущее время. И поддержка есть у современных Mozilla, Opera и браузеров от Google. Однако, и это не мало.

Тема web-приложений в режиме offline существует давно. Возможно ServiceWorker и CacheStorage лишь временное решение. Тем не менее, это уже имеет смысл использовать. Даже если данные технологии устареют, производители браузеров создадут что-нибудь в качестве замены. Стоит только следить за прогрессом в этом направлении!

https://habr.com/ru/post/279291/

# 49.Работа с базами данных.

Ответ:

Хранение данных и их обработка в приложении - неотъемлемая часть

полноценной работы приложения.

Базы данных отвечают на вопрос, где хранить информацию и откуда ее брать, чтобы обеспечить функциональность приложения.

Работу с базами данных можно разделить на:

*Локальное взаимодействие с БД*; *Сетевое взаимодействие с БД;*

*Локальное взаимодействие с БД*;

Локальное хранение данных производится на жестком диске устройства. Это имеет свои плюсы, например доступность к данным в любой момент времени, а так же и минусы, например при повреждении жесткого диска потеряются все файлы.

В качестве реализации хранения данных в локальном хранилище можно привести пример, использования БД в андрод приложении, локально.

Это более сложная и многогранная система работы с данными - реализация реляционной базы данных.

Базы данных позволяют структурировано хранить данные приложения, а также получать необходимые данные в информативном объеме, благодаря запросам.

Локальная база данных всегда доступно для обработки данных, файлы сохраненные в приложении будут доступны только ему, а при удалении приложения с устройства данные удалятся с ним.

“Родным” для приложения Android является реляционная база данных на основе SQLite.

*Сетевое взаимодействие с БД;*

Сетевое хранилище данных более функциональное и что самое главное надежное, по сравнению с локальным.

В локальном хранилище все данные хранятся на 1 устройстве. Потенциальная поломка жесткого диска (в локальном хранилище) или внешнего хранилища угрожает лишиться всех данных приложения.

Сетевое хранилище данных позволяет реализовать взаимодействие приложения с базой данных на сервере благодаря запросам: get, post, put, delete.

Пример взаимодействия:

Один из наиболее удобных способов передачи данных с серверным хранилищем реализуется с помощью файлов формата json.

# 50. Технологии создания нативных приложений на веб-стеке.

Ответ:

Нативные приложения — это приложения, которые разрабатываются для конкретной платформы при помощи соответствующего этой платформе языка. Например, для создания приложений, которые будут работать на iOS, потребуется знание Objective-C или Swift. Разработчики для Android должны знать Java или Kotlin.

При разработке нативных приложений используется определённый набор инструментов: комплект разработки программного обеспечения (SDK — Software Development Kit) и интегрированная среда разработки (IDE — Integrated Development Environment).

Нативные мобильные приложения ориентированы на конкретную операционную систему, поэтому более безопасны, интуитивно понятны, лучше работают, предоставляют разработчикам полный доступ к функциям целевого устройства (GPS, камера, телефон и т.д.). При этом создание нативных приложений требует больше времени и финансовых затрат.

Несмотря на более затратную разработку, основная часть заказов приходится именно на нативные продукты. Используя нативные технологии, можно реализовать практически всё, что хочет заказчик. Приложения получаются технически адекватными и способными решать максимальное количество задач.

Плюсы:

* Превосходная производительность: нативные приложения выполняются плавно и без зависаний даже в случаях повышенной нагрузки на графический процессор и интеграции сложных вычислений.
* Наличие доступа к индивидуальным возможностям платформы: самое лучшее в этом типе приложений  —  это то, что они обеспечивают доступ к встроенным возможностям устройств или конкретной платформы.
* Нативный пользовательский интерфейс: плавный опыт использования обеспечивается благодаря тому, что приложения создаются в соответствии со стандартами платформы.

Несомненно, нативные мобильные приложения по производительности не имеют равных. Однако, помимо чистой мощности и практически безграничных возможностей, нативные приложения отличаются высокой стоимостью разработки.

Кроссплатформенные или гибридные приложения дешевле.Они создаются с единой базой кода, допускающей их запуск на нескольких операционных платформах. Хотя этот вариант менее мощный и имеет ряд ограничений в функциональности, с точки зрения разработки он более быстрый и экономичный (стоимость разработки может быть в два раза ниже создания нативного приложения).

Типичные представители этого семейства: React Native, Native Script, Electron.

Комплект разработки программного обеспечения (SDK) позволяет разработчикам использовать для создания приложений предпочтительные языки программирования. При этом SDK поддерживают интерфейсы прикладного программирования (API), которые можно использовать на языках программирования, изначально несовместимых с поставщиком ОС.