Всем привет, добро пожаловать. Этим стулом GAMS начинает новую серию полукресел, а мы этим видео начинаем новую серию видео, она будет называться «Читаем Луца». Когда человек, который начинает учить Python, спрашивает, что же мне почитать, наверное, процентов 90 разработчиков говорят «Читай Луца, там все есть, это такой огромный двухтомник, он жутко нудный, он очень подробный, он достаточно плохо переведен, иногда он очень косноязычен». Но, к сожалению, пожалуй, лучше, чем… Этот учебник просто по базовому Python пока ничего нет. Мы будем читать это дело по главам, на каждую главу у нас будет, скорее всего, отдельный оратор, и он будет давать презентацию, а по ходу вот этой презентации я буду задавать вопросы, каверзные и не очень, ну или подкидывать тему для обсуждения. Мы тут не одни зрители, которые безмолвно присутствуют у нас. Сейчас в чате они тоже будут задавать вопросы, мы их будем или зачитывать в конце, или мы будем в процессе обсуждения с ними как-то подключаться. Сегодня на первую главу у нас будет читать Алексей Троянов. Привет, Алексей. Добрый день. Ну что, я не буду утомлять тебя софтами. Поехали, наверное, да? Включаю демонстрацию экрана? Да, совершенно так. Отлично. Добрый день, меня зовут Алексей, я студент факультета Backend Python, 27-я когорта. Сегодня предлагаю обсудить первую главу замечательной книги Марка Лутца «Изучаем Python». Что? Обсудим сегодня достоинства и недостатки Python, технические превосходства по сравнению с другими языками программирования. Так, погоди минуту. Сразу некоторые вопросы у меня есть, да? Я обратил внимание, что там пятое издание. Пятое издание, оно вышло на английском. Это оно на русском вышло в 19-м году, в 19-м, 20-м. А на английском оно вышло в 2013-м. Вот, это очень важно, это нужно понимать. Издание рассчитано на версию языка Python. Там даже 2.7 был, но мы его как-то, наверное, не будем, да? Оно рассчитано на Python 3.3. Что это значит, как по-твоему? Какие недостатки? То есть нам нужно понять границу применимости наших знаний. Которые мы сейчас будем обсуждать. Я не могу сходу сказать разницу. Ага, ну смотри, то есть если говорить про сейчас вот уже 3.10, мы все-таки живем в 22-м. Ну, очевидно, что какие-то фичи добавлены. Это совершенно точно. Я вот, например, навскидку могу сказать про TypeHint, про дата классы. Про какие-нибудь там кучу сахара, тому, что у нас нужно объединять словари, например, новыми вот этими вещами. Морж-оператор, иниты в пакетах и прочее. В принципе, для первой главы не критично, но для первой главы критично может быть вот что. Когда мы сравниваем это дело с какими-то другими языками, вполне возможно, что ситуация... И вот эти граничные условия нам нужно иметь в виду, когда мы читаем эту книжку. Книжка хорошая, но у нее есть некоторые моменты просто из-за того, что лутц не может. Ну, это так отскочим в сторону. Давай дальше. Итак, почему же люди используют Python? В наши дни, естественно, имеется огромное количество языков программирования. Во-первых, это качество программного обеспечения. Пишем код мы один раз, а перечитываем его или используем многократно. Поэтому по замыслу Python должен быть читабельным, а значит, сопровождение его намного легче, чем традиционных языков. Тем более при соблюдении PEP-8 он понятен, даже если написан не вами, кем-то сторонним. Второе. Продуктивность труда разработчика. Код Python более краток и лаконичен. Аналогичный код Python по сравнению с плюсами или Java примерно на четверть короче. А это не только меньшие затраты при наборе кода, но и при отладке или сопровождении. Третье. Почему выбирают Python? Это переносимость программ. То есть кроссоверность. Кроссплатформенность. В большинстве случаев программы, написанные на одной платформе, достаточно просто скопировать на другую операционную систему, где установлен Python. Четвертая причина. Это, соответственно, большое количество поддерживающих библиотек. В стандартной библиотеке входит большое количество различных расширений для различных прикладных задач. Кроме того, существуют библиотеки сторонних разрешений. У работчиков также охватывающие все сферы прикладного программирования. От веб-сайтов конструирования до низкоуровневого доступа к железу. Еще одна из причин – это интеграция компонентов. Python позволяет легко обращаться к библиотекам и взаимодействовать с другими частями программы, которые написаны на других языках. А вот здесь вопрос. Погоди. Отлично, что коснулся. А каким образом вообще это можно сделать? Как ты себе это представляешь на практике? Библиотекам обращаться. То есть мы импортируем библиотеку и, соответственно, ее используем у нас. Да, правильно я понимаю? Да. Можно ведь, наверное, еще другие какие-то варианты? Какие есть мысли? Мы можем, например, код на плюсах взять и засунуть в питаньячий файл? Или не можем? Скорее всего, можем, только я не пользовался такой хищникой. Окей, ладно. Ну смотри. Дальше как бы у нас есть… На самом деле я вот себе вижу три варианта, как мы все это можем сделать. То есть мы, например, запускаем программу на плюсах. Она что-то там делает у себя, да, потом вызывает, импортирует у себя скрипт в питон, да, и что-то там ему отдает. То есть фактически весь код питона мы можем запускать, под контролем кода на C+, ну на плюсах на C++. Второй вариант – это что мы можем действительно взять и вызвать какую-нибудь функцию из модуля библиотеки, вот когда мы исполняем питон, и он плюс работает внутри питонного. Ну а третий вариант, мне кажется, что есть, например, две независимые программы, одна на плюсах, другая на питон, ну и они каким-то образом обмениваются информацией. То есть там общая память, может быть, каналы, сигналы, азбука Морзе, REST, все что угодно, но они работают независимо и, соответственно, между собой обмениваются. Как по-твоему, какие вообще варианты из этих возможны? Ну, мне кажется, первый и второй – это точно библиотеки и вызов кода. Ну, в принципе, обмен через какой-то стандартный интерфейс, там REST это или что-то другое, вполне возможно. Возможно, почему нет. То есть, получается, три варианта, как мы можем интегрировать. Окей, давай про наслаждение программированием поговорим. Что для тебя лично наслаждение программированием? В чем кайф? Кайф? Ну, во-первых, если говорить простым языком, это заставить железяку сделать то, что тебе нужно. А с другой стороны, это должно быть увлекательно и приятно, и легко, а не погружаясь там. То есть, у тебя ломает там обрабатывать ошибки, управлять памятью и все, вот такую вещь. Если это как бы интуитивно понятно и легко обрабатывается, почему бы нет? А если для этого нужно там в ассемблер лезть, то это, наверное, сложновато. Ну, отлично. Давай поедем дальше. Поедем дальше. Так, о... Автор задается вопросом, является ли питон языком написания сценариев. Ох ты. А что такое вообще язык написания сценариев? Ну вот, как раз в том тексте, который я сейчас написал для этого слайда и озвучиваю, как раз это и будет рассматриваться. То есть, ну давай по-простому. То есть, как ты себе это понимаешь? То есть, что такое язык написания сценариев? Не человеком, не луцевским языком, от которого мы как раз хотим отойти. А вот как, на твой взгляд, вот это все... Для меня ближе язык написания сценариев, это вот первый вариант, который на слайде есть, это инструменты командной оболочки. То есть, сценарии для управления системой, какие-то скрипты, утилитки и подобное. Слушай, ну логично, потому что в Unix-системе он, например, прямо встроен. И, в общем-то, написать на нативном языке что-то, что работает с самой системой, ну, наверное, окей. Насколько я помню, еще Raspberry Pi очень здорово это дело любит. Там, в общем, Python тоже нативный язык, и это круто, потому что мы, получается, под себя гребем целый кусок интернета вещей. Логично? Да, абсолютно. Вот. Окей, я смотрю, у тебя есть процедурно-объектно-ориентированная функциональность. Ориентированное программирование ты выделил, а в чем между ними разница вообще? Ну, соответственно, это три парадигмы программирования. Процедурное программирование – это стандартное программирование, которое было основополагающим, соответственно, последовательным выполнением каких-то там простых действий. Потом пришло объектно-ориентированное программирование, где стали рассматриваться конструкции. Конкретные объекты и подходы работы с ними, такие как инкапсуляция, наследование, полиморфизм. Соответственно, по прошествии какого-то времени подошли к функциональному программированию. В нем отсутствуют операторы условия, циклы. Все используется в основном функции, которые могут вкладываться друг в друга, обертываться. И так далее. Окей, ладно, хорошо. Но не вполне себе уверен, что это правильная формулировка, если честно. Я надеюсь, что ребята, которые нас будут смотреть, они имеют что сказать по этому вопросу. То есть, в общем-то, три парадигмы и разница между ними – это интересно, это познавательно. Это очень классно. То есть, вопрос возникает номер раз по точности формулировки, и вопрос номер два – по времени возникновения. То есть, возникло ли функциональное программирование до или после ООП. Предлагаю пока на этот вопрос не отвечать, посмотреть, что напишут нам наши замечательные зрители. Я почти уверен, что у них есть что сказать. Давай дальше посмотрим. Переходим к недостаткам по этому. Ага. Да, давай по-простому. Да, то есть, как бы знаю, что ты подготовился, прочитал текст. То есть, кайф заключается в том, что у нас, у Питона есть, на нем писать быстро, но бесплатного ничего не бывает. Да. То есть, в чем минус? За удобство. За удобство надо оплатить скорость. Конечно, конечно. Да. Почему частота внесения изменений – это минус по-твоему? Наоборот, классно, мне кажется, язык развивается, придумываются новые фичи. Да, но возникает вопрос с поддержкой собственных ранее написанных программ. Соответственно, нужно отслеживать обратную совместимость своего кода. Ну, в чем проблема? В чем проблема, как бы, если ты пишешь… Ну, просто более новые программы, они используют старые фичи. То есть, старые фичи тоже можно использовать. То есть, ну, например, та же самая строка, которая, не помню, в 3.6, по-моему, или в 3.7, в 3.6, по-моему, Питоне появилась. Ну, до этого был формат. Но в 10, например, формат тоже используется без проблем. То есть, можно форматировать строчку другими методами. То есть, обратная совместимость. Мне кажется, все-таки частота внесения изменений – тут в чем проблема больше. В том, что если ты, например, читаешь курс по Питону, ты такой, ааа, и тебе из зала такие, а это уже делается по-другому, чувак, ты чего? Для преподавателей в основном, да. Да, мне кажется, это вот у Луца, он это в минус сделал, это он просто про себя думает, что ж такое-то каждый год книжку писать. Это я улучшусь, кто же ее купит. Он не говорит про отрицательные стороны, он говорит именно про компромиссы, которые возникают. Возникают из-за тех или иных событий. Ну окей, но видишь, у тебя слайд недостатки Питона, поэтому я подумал, что… А, у тебя недостатки – это скорость выполнения, а это компромиссы. Окей. Слушай, а батарейка в комплекте, что это значит? Ну, с точки зрения Луца, батарейка в комплекте – это когда все включено, что с Питоном поставляется большое количество библиотек, инструментов. И иногда привязывание к тому или иному инструменту. И со временем может как бы… Ну, ты его можешь перерасти, а поддерживать… Бывает, что, как он говорит, батарейка изменяется или повреждается, или устаревает. Соответственно, можно поддерживать этот код свой. Слушай, ну, блин, проблема Legacy, она, в общем-то, характерна для программирования. У меня есть ребята, которые работают в Яндексе. Правда, они с этого проекта уже ушли. У которых там в одном месте вообще джанго, не соврать, 1.3. 1.3, что ли, 1.4. Просто потому, что к ней приплели стороннюю библиотеку, которая как-то очень слабо обновляется, и ничего не получается. В общем, с одной стороны, оно работает, с другой стороны, как-то развивать это сложно. С третьей стороны, ну, блин, работать не трогай. И деньги на рефакторинг как-то… Ну, не нашлось и ресурсов, ни денег. Возможно, сильно… Версия будет дороже себе. Согласен. То есть, как бы здесь… Ну, мне кажется, эта проблема, в общем, даже не питона, а проблема самого программирования. Слушай, а недостатки для проектов с открытым кодом – это ты про что? Что ты имеешь в виду? Ну, вот ПО с открытым кодом бывает больше иногда похоже на анархию, как пишет Луц. И ссылается на то, что, несмотря на выстроенный процесс контроля ошибок, иногда в релизы попадают… Некоторые там ошибки нерабочие. Например, в версии Python 3.2.0 поступила с неработающей консольной функцией input. Окей, обрати внимание, что отличный свежий пример 3.2.0. Ну, понятно, что… Актуально было, наверное, году… В 30-м году. В 10-м, наверное. В 10-м, ну… Дай бог. Вот, окей. Ну, смотри. На самом деле, вот по скорости выполнения… Чатики пишут. В общем-то, ну, такое на самом деле себе. То есть, потому что какие-то вот bottlenecks, места, где есть проблемы, есть критичные вещи по поводу того, что нужно быстро. Но мы можем кусок этот на плюсах переписать и взять эту библиотечку использовать для вычислений. То же самое NumPy, да, то же самое Луц, он, в общем-то, на него ссылается. Он очень хорошо умеет и математику, и быстро. Отличный вариант, то есть, ну, такое. Да, я хочу сказать, я извиняюсь, что слайд по недостаткам питона, поэтому нужно было что-то написать, нельзя написать, что у него нет недостатков. Блин, ну… Понятно, что дальше как раз на следующих слайдах будет говориться о том, что все узкие места можно переписать на C, C++, и будет вам счастье. Ну, да. Ну, например, вот народ пишет, что… В 3.9 утечка памяти обнаружилась. Вполне возможно, что наши любимые зрители могут еще какие-нибудь баги, косяки, с которыми они столкнулись в работе с питоном, в общем-то, написать в комментариях. С большим удовольствием почитаю. Это, в общем-то, хейт питона, это наше все, это интересно. Слушай, ну, а вот если говорить, просто я не знаю твоих следующих слайдов, там будет про пеп? Нет, про пеп нет. Отлично, самое время поговорить про пеп. Ну, вот смотри, то есть, по сути, вот эти вот проблемы, они, по большому счету, лечатся тем, что мы используем пеп. То есть, все изменения сначала тщательно обкатываются, должны, ну, вот в идеале, и потом только обсуждаться комьюнити, да, и потом только приниматься или не приниматься. Ну, кажется, что… Лучшего варианта вообще нельзя придумать. То есть, открытый код, закрытый код, какая разница? Ну, одним еще из недостатков про открытый код, он говорит о том, что коммерческая выгода в закрытом коде, в коммерческом проекте обменивается на личные предпочтения текущего круга разработчиков, которые могут либо совпадать, либо не совпадать с вашими предпочтениями, с нашими программистами. Ну, тут такая… Такое дело, ну, как бы, есть специально обученный великодушный диктатор, я надеюсь, что гвида нам поможет. Ладно, давай дальше посмотрим, что, какие следующие слайды ты еще подготовил. Йо-хоу, системное программирование. Давай прямо коротенько поговорим про вот это вот все. Ну, системное программирование мы с тобой обсуждали, да, это же у нас всякие вот… Интерфейсы, те утилитки, которые позволяют работать с системой. То есть, ну, графические интерфейсы, это у него тоже какой-то, мне кажется, в книжке там какая-то старость написана, я почти уверен, что есть и более современные фреймворки. Будет классно, кстати, если ребят напишут в комментариях, какими вот, какими фреймворками и какими вещами они используются, то есть для графических интерфейсов, для интернета. Для интеграции компонентов, для баз данных, для, ну, вот, прототипирования, это такая вещь, то есть, в общем-то, здесь, наверное, это как-то более объединенное, да. Ну, наука, это, скорее всего, MyPy какой-нибудь, то есть, наверное, ничего нового не придумали, хотя будет классно, если вы нам расскажете. Ну, игры, роботы и таблицы Excel и всякое вот это вот, там вообще очень большой пласт. И здесь, конечно, лучше он… Немножко устарел, потому что здесь можно добавить еще и компьютерное зрение, распознавание образов и вот это вот… Искусственный интеллект, да, то есть… Да, да, да, то есть, как бы, machine learning, вот это вот, всю вот эту вот бодягу можно, можно, в общем-то, сделать. Вот народ накидывает, ухоу, киви, кьюти, тинктер, SQL, OpenExcel, OpenPyExcel, нифига себе. Кинтер как раз у Луцы и указывается. А, слушай, ну, видимо… Видимо, кто-то пережил. Я у него посмотрел фреймворки, которые он рассказывает про веб. Честно говоря, я только Django знаю, но, наверное, остальные куда-то канули в лето, и почему-то нету у него фласка, непонятно. То есть, по-хорошему, здесь можно взять и накидать… У него, кстати, этого фастапи тоже нет. Вот, взять и накидать всяких интересных названий. Вот, взять и накидать всяких интересных названий, которые вообще часто используются и используются в вашей жизни. Ребят, не стесняйтесь, давайте сделаем книжку Луца еще лучше и еще интереснее тем, что она будет актуальная. Да, еще тортой съесть, вот, для баз данных, я точно знаю. Поехали дальше, посмотрим, что еще. Вау. Ну, соответственно, недостатки посмотрели. Технические превосходства. Прекрасно, давай поговорим про технические. Объектно-ориентированный, функциональный мы уже разобрали чуть раньше. Соответственно, в книжке говорится о том, что он первоначально был… Потерялся сейчас секунду. В общем, он легкий скриптовый язык. Плюс подключение. Подключены к нему объектно-ориентированные фишки. Плюс добавилась функциональность. Ну, то есть он сначала был процедурщиной, да? То есть он был процедурным языком, а потом на него всякое нарастили. Окей, бесплатно – это круто. Слушай, интересно, есть вообще нормальные платные языки по подписке, прикинь? То есть ты можешь программировать по подписке каждый месяц за то, что ты программируешь. Да, на чудесном языке ты должен бабла платить. Я, честно говоря… Мне сложно сейчас представить. Тут, наверное, все-таки говорится о открытых и закрытых. В том смысле, что тот же самый Visual Basic, он пропиетарный. Те, которые разрабатываются закрытыми коммерческими предприятиями, компаниями. Гошечка, например, да? А, слушай, нам нельзя рекламировать продукты этой фирмы. Мы не рекламируем, мы просто говорим, что есть такой язык. Не пользуйтесь им, это плохой язык. Ну, или пользуйтесь, но на свой страх и риск. Вы должны понимать и осознасить. Слушай, переносимый, то есть есть непереносимый… Непереносимый, ну, это такая сложность перевода, соответственно, это о кроссплатформенности. Ну да, то есть, наверное, у нас переводчик не программист это все дело переводил, поэтому действительно он будет исполнять. Хотя, насколько я помню, для винды там нам и интерпретатор все-таки надо ставить. Ну да, поэтому ставится интерпретатор. Да, он из коробки в винде не идет, ну, как бы, ладно, окей. Мощный, слушай, динамическая типизация, честно говоря, здесь он прям вообще попал не туда, потому что сейчас питон со всей мочи скачет в сторону тайп-хитинга, вот этой вот аннотации типов, потому что, чтобы, видимо, работать быстрее, наверное, так. Потому что аннотация типов не отменяет же динамику. Динамическая типизация. Нет, она не отменяет, но получается какая-нибудь интересная штука, на двух стульях сидим, то есть, с одной стороны, мы как бы типизируем динамически, но, в общем-то, мы при инициализации, мы явно задаем класс и, в общем-то, сильно ругаемся, если это все, задаем тип и сильно ругаемся, если посредством MyPy, если у нас этот вот тип, он как-то... Не совпадает. Не зашел. Да, не зашел. Слушай, автоматическое управление памяти, блин, ну, тут тоже много копий сломано, такое дело, окей. Да, сборщика мусора. Да, то есть, гербочколлектор, ну, в принципе, у него неплохой, насколько я могу судить, да, и это удобнее, чем руками. Но, тем не менее, иногда он и лажает. Я помню, что кто-то даже отключал в каких-то проектах, что-то там читал. Да, вот бестификат. И степизация как-то, ну, достаточно сложная и неудобная. Так что есть ощущение, что они раскаялись и постепенно ведут в сторону, ну, путем таким вот хитрым введением тайп-хинтов. Слушай, что такое программирование в большом? Ну, это как раз о том, что он является как бы надстройкой над другими, может являться надстройкой над другими языками, интегрирует в себе... Скриптовый легкий язык, с которого можно вызывать там C, C++, другие языки программирования. Окей, то есть, вот это программирование в большом, то есть, вот, по сути, это клей, то есть, он может быть такой универсальной прокладкой между джабой, да, то есть, и вот это все. Окей, встроенные типы объектов, это речь идет про всякие там листы, дикты и вот это все, да? Да, да, абсолютно. Отлично. Инструменты? Да. О том, что они изменяются динамически в процессе и легко встроены? Да, что с ними удобно работать, я согласен. В общем-то, у Python достаточно приличный вот этот вот кусок для работы с данными, прям все хорошо. Ну, инструментики, это речь... О встроенных библиотеках. Ну, да, то есть, как бы достаточно много на все случаи жизни. Те самые батарейки... Батарейки, о которых так мы немножко ругались, но на самом деле они как раз дают как раз мощь. Да, радости к программированию. Согласен, та самая радость. Так, ладно, утилиты, это... Это сторонник разработчиков, соответственно, то, что большое количество инструментов... Ну, то есть, сел и написал, то есть, по сути... Выложил на гид, пользуйтесь. Ну, логично, он же, в общем-то, открытый язык, почему нет? Окей, смешиваемый. С кем? А, слушай, смешать, но не взбалтывать. Или, наоборот, взболтать, но не смешивать. Не помню, как там правильно. Ну, это как раз, наверное, повтор про программирование в большом, о том, что самыми разнообразными способами можно вызывать IP-интерфейс на C, имеющийся в Python, и, наоборот, там, как бы... Ну, да, слушай, ну, возможно. Не вызывать, да. Да, так, ну, простой в использовании. И изучение, все признается в сравнении. Да, относительно. Basic намного проще, а уж Pascal-то, ба... Ну, у Basic есть другие недостатки. Да, слушай, ну, везде, то есть, к ним любой язычок можно взять и недостатков-то доковырять. Слушай, группа Мойти Пайтон, ты что-нибудь смотрел из нее? Узнал только, когда начал приобщаться к Python, не смотрел ничего. Такие. Поверхностные знания. Ясно. Ну, там, знаешь, такой интересный английский юмор, достаточно специфический. Слушай, это не сериал, это группа как раз комиков. Комик-группа. Да, и они делали в том числе и сериал, там, про Грааль что-то. Кстати, там очень классный, он очень легкий язык, сильно рекомендую его для изучения английского. Ну, например, с подстрочником, у них хорошие произношения, мне очень нравится. Теперь, внимание, вопрос. Ну, назван он в честь группы Мойти Пайтон, а почему на логотипе у него питон? Возможно, чтобы как-то зашифровать это, чтобы не серьезно как-то в честь группы Мойти Пайтон комиков называть. Да нет, слушай, на самом деле там интересная история. Я погуглил КС Грешин, но это было давно, потому что мне тоже было интересно, почему Пайтон, а вообще питон. Просто Гвида, он тот еще... Он еще раздолбай оказался, ему, в общем-то, было все равно на нейминг абсолютно. И когда ему нужно было придумать язык, там что-то там с ABC у них было, он взял и сказал, ну, пусть будет Пайтон в честь Мойти Пайтона. Вот. А логотип просто взял первый попавшийся шрифт и написал слово питон. Все. Больше ничего. Но в итоге язык начал немножко стать популярным. И люди, которые там писали книжки, еще что-то, начали извращаться и рисовать всяких змей на обложках. Там на журналах, на сайтах и везде. То есть шрифт, в общем-то, мало кто пользовался. Если посмотрите Вику, то, в общем-то, можно понять, почему. Он такой страшненький, из шариков состоит. Он ужасен на самом деле. Настолько пипец. Почему у него крыса нарисована, а не питон? Ну, потому что, наверное, питон ест крыс. Это логично. И вот они в 2006-м придумали логотип как раз с двумя желтыми-синими питонами. Ну, и обновленный шрифт. То есть просто люди заставили взять и все-таки... Люди проголосовали не за Мойти Пайтона, а за змей. За ребрендинг. За ребрендинг, да. Ну что, слушай, звучит как тост. Ну что, за ребрендинг. Как тост. Это у тебя последний слайд. Еще один. Оу-хоу. Так, ладно, смотрите. Здесь, наблюдая я много вымерших языков... Ну, абсолютно, да, если книжка-то старенькая. Вот. Ну, как бы, Перл... Кстати, интересно, на Перле кто-нибудь пишет? Прямо вот активно сейчас. Я не уверен. Это можно посмотреть в этом... Как он называется-то? Есть сравнение языков программирования по популярности. Ой, слушай. Там достаточно столь... Странные у них метрики, то есть там они меряются, например, количеством запросов, еще что-то, количеством реп в гите на этом языке. Я бы не стал прямо вот говорить, что это вот прям вот так, вот так. Так, друзья мои, кто нас смотрит, напишите языки, на которых вы разрабатываете. Будет очень классно, если найдутся люди, которые пишут на Алоа. На Перле, на ТСЛ. Это будет круто. Это будет вообще не мейнстрим ни разу. Это будет вообще очень круто. Вот, Visual Basic тоже, это уважаемо. Вопросов нет. Остальные вроде ничего так. Ну, давай как-то мы, наверное, будем... Я для работы поописываю иногда на Visual Basic. А, ну видишь. По необходимости. Слушай, а что Visual Basic делает? Ну, он просто в... Также можно скриптовым его использовать в ВБС для винды. Ну, окей. Ладно, это очень круто. Так, ну давай стопшеринг и начнем подводить итоги. Задаем вопросы, но вопросы не из книжки. Вопросы из книжки были бы слишком простые. Что скажешь? В чате, что работает на СЛО. Ну, в общем-то, круто. Слушай, оказывается, нет. Есть у нас, среди нас еще люди, которые умеют, любят и практикуют всякое. Итого, питон нормальный, как по-твоему? Лучший. У меня последний слайд не показал. Там написано, что он best of the best of the best. Слушай, это нельзя, потому что это с точки зрения... Это нарушение... Это нарушает закон о рекламе. Боюсь, что нас могут взять и... Взять и это самое. Окей, народ пишет, что есть параграф H, строка А. Вопрос. Кошмар. Я не буду это все делать. То есть, друзья мои, изучайте питон. Питон классный, питон современный. Питон развивается. Читайте Лутца, слушайте Лутца вместе с нами. Мы постараемся... Мы постараемся копать немножко глубже, чем Лутц, потому что книжка написана немножко давно. И в следующих сериях мы обязательно будем, в общем-то, спрашивать себя, а как по-современному мы будем смотреть на дедушку... Как, кстати, у него имя? Марк. Вот. На дедушку Марка. Вот. И, соответственно, немножко помогать ему сделать книжку актуальной. Дальнее, внося новые современные модные техники. Ну что, а на сегодня это все. Спасибо большое, что пришел. Было очень круто. Мне понравилось. Динамично. Тебе тоже понравилось. Ну что, живите долго и процветайте. До свидания. Продолжение следует. Хаудихо, друзья! В комментариях все уже давно просят сделать ролик про ООП и объяснить, что же это такое и для чего оно вообще нужно. Так что давайте познакомимся с, казалось бы, такой страшной штукой, как ООП. Хотя на самом деле в этом нет ничего сложного, потому что ООП было придумано лишь для того, чтобы облегчить жизнь программистам, но никак не наоборот. И если вы вспомните, как начинали учиться программировать, то вас скорее всего учили писать код в процедурном стиле. Это значит, что код, который вы писали, выполнялся последовательно и состоял преимущественно из всевозможных подпрограмм или же обычных функций. Так вот, до изобретения ООП таким образом код писали по сути все. И прикол тут в том, что пока ваша программа маленькая, процедурный стиль никак не мешает ее поддерживать. Или как-то модифицировать. То есть в коде условно всего 200-300 строчек кода, который легко читается и запутаться в нем по сути нельзя. Но программисты очень быстро столкнулись с тем, что программы стали большими. И вот когда в такой условной программе где-то 10 тысяч строк кода, то читать такой код становится уже невозможно. И даже если сильно постараться все хорошо закомментировать и попробовать разбить этот код на файлы, все равно останется масса проблем. И вот для решения всей этой каши было придумано ООП, или же объектно ориентированное программирование. Основная идея которого лежит в основе того, что все состоит из объектов. В отличии, например, от того же процедурного программирования, где каждая программа это один большой кусок кода, состоящий из множества переменных блоков и функций. То есть любая объектно ориентированная программа, программа это всегда набор компонентов взаимодействующих друг с другом и все они называются объектами при этом здесь важно понимать то что программист создает объекты таким образом чтобы они все участвовали в выполнении задачи но при этом каждый из них был бы обособленным то есть каждый объект в программе отвечает только за себя и является полностью автономным самостоятельным компонентом дальше каждый такой объект может быть представлением чего-то реального например клиент магазина или товар которые могут являться частью целого интернет магазина либо объект может представлять что-то абстрактное что существует только в компьютере условно это может быть какая-то структура данных . на экране или что-то в таком духе но для программы это все не более чем просто набор объектов каждый объект может быть объектом который в этом доме или что-то в таком духе но при этом каждый объект может быть объектом который из которых общается с программой и взаимодействует с другими объектами посредством своего публичного интерфейса а вот вся его внутренняя реализация она полностью скрыта и такой механизм называется инкапсуляции то есть реализация объекта и его данные скрываются от других компонентов это бывает полезно тогда когда к примеру доступ к какой-то переменной нужно предоставлять не напрямую а с помощью специальных методов для выполнения и довольно-таки специальных методов для чтения и записи. Да и вообще, если хорошо продумать и спроектировать всю программу, то благодаря инкапсуляции можно будет работать над каждой ее частью отдельно, не охватывая умом всю программу целиком. И еще в ООП есть такая штука, благодаря которой можно легко создавать огромные объекты, разбивая их на несколько маленьких. Другими словами, такие объекты могут собираться сразу из нескольких других, выступая своего рода агрегатором реализации. Например, можно взять два объекта, компьютер и монитор. Каждый из них реализует какой-то свой функционал. И дальше можно создать большой объект лэптоп, состоящий из более маленьких объектов компьютер и монитор. И такой принцип дает возможность сильно упростить сложность всей программы, а называется он наследование. Это еще один мощный инструмент объектно-ориентированного программирования. Все это позволяет создавать очень большие программы, над которыми могут работать сразу сотни или десятки тысяч программистов. Но при этом все они могут не знать, как устроена вся программа целиком, потому что каждый из них будет работать лишь над ее отдельными частями. Ну и третий мощный инструмент ООП, который пока что можно пропустить, это полиморфизм. Потому что понимается он гораздо лучше тогда, когда вы уже изучите концепции ООП на практике. Пока что можете запомнить лишь то, что полиморфизм это когда функция способна обработать данные разных типов. Ну и по сути это все инструменты, которые имеются в ООП. Их всего три, а именно инкапсуляция, наследование и полиморфизм. И как видите, ничего сложного в этом нет. Соответственно, сегодня практически все языки в ООП имеют полиморфизм. И если вы хотите узнать больше о полиморфизме, то смотрите это видео до конца. В этом видео я расскажу о том, что полиморфизм и полиморфизм являются объектно-ориентированными. Сюда относятся и Python, и Java, и PHP, и C++, и многие другие. Так что понимая общий принцип работы ООП, вы заложите в себе фундамент знаний любого объектно-ориентированного языка программирования. И особенно это касается языка Python, потому что хоть он и допускает различные методологии программирования, в его основе лежит все-таки ООП. Поэтому абсолютно все в Python. Если вы хотите узнать больше о полиморфизме и полиморфизме, то смотрите это видео до конца. И если вы хотите узнать больше о полиморфизме, то смотрите это видео до конца. Ну а если вы еще только начинаете изучать Python или планируете это сделать, то порекомендую обратить внимание на курс Python для веб-разработки. Занятия здесь будут длиться 9 месяцев, и это как раз то время, за которое вас нехило натаскают и вы действительно научитесь программировать. В том числе большое количество времени учебы уделяется объектно-ориентированному программированию. А походу, это не только для вас, а для всех, кто не знает, что это. А это очень важный момент, ходу учебы выдаются домашние задания и создается настоящий проект. Ну а для записи на курс вам не нужны никакие стартовые знания, всему научат на месте. Все, что от вас требуется, это желание стать Python-разработчиком. В общем, все, что нужно для качественного старта. Ну а ссылку на курс я оставлю в описании под роликом. Сообщите менеджеру промокод Howdyho и получите 35% скидку до 20 мая. Окей, сейчас вам может показаться, что вы хорошо поняли, что такое ООП и знаете его основные преимущества. И это, конечно, хорошо, но пока еще рано переходить к практике. До этого вы обязательно должны понять, что такое класс и объект. И это действительно важно. Настолько важно, что я даже скажу такую штуку. Если вы четко поймете, что такое класс и объект, то можете смело считать, что вы знаете как минимум половину всего ООП. И я сейчас не шучу. Так вот, объяснить, что такое класс, можно в принципе по-разному. Но если делать это совсем по-простому, то класс это просто кусок кода, у которого есть какое-то имя. И чтобы воспользоваться этим куском кода, нужно из этого класса создать объект. Эту процедуру еще называют инициализацией класса. Или созданием экземпляра класса. Ну или просто созданием объекта. Как не все одно и то же. И такой объект принято называть экземпляром класса. Но не будем пока забегать так далеко. Чтобы вы лучше понимали, давайте представим себе какой-то объект из реальной жизни. Ведь, как вы помните, концепция ООП подразумевает, что все является объектами. Соответственно, вокруг нас очень много разных объектов. Это стол, стул, дом, собака, кошка, другие люди, ручка, и так далее. Для примера давайте возьмем автомобиль. И подумаем, что нужно, чтобы создать автомобиль. В первую очередь это, конечно же, чертеж, в котором будет описано, как и из чего собирается авто. Ну и там же будет описан полный список нужных деталей. Так вот, если для того, чтобы собрать машину, нам нужен чертеж, то в программировании для того, чтобы создать объект, нам нужен класс. И думаю, вы уже начинаете понимать, как это работает. Соответственно, класс — это тоже своего рода чертеж, в котором довольно точно перечислено, из чего он состоит и какой реализует функционал. Ну и так как в жизни мы можем из одного чертежа создать сколько угодно автомобилей, единственный наш лимит — это количество оставшихся деталей или же ресурсов, в программировании это устроено похожим образом. Из одного класса можно создать сколько угодно объектов. А ресурсом будет выставка. И в этом случае будет выступать доступная мощность компьютера. То есть, чем больше мы захотим создать объектов, тем больше потребуется памяти. И для примера давайте на Python создадим какой-то простенький класс в качестве небольшой практики. Пусть этот класс будет называться color, и в нем будет ровно три переменные, а именно red, green и blue. И как вы уже понимаете, мы создали класс для репрезентации цветовой системы RGB. Это уже не какие-то аналогии, а вполне возможности. И это вполне реальный класс, который особенно часто встречается в игровых движках или библиотеках для работы с графикой. В таких классах обычно существует метод для перевода RGB в HEX-репрезентацию цвета, и в нашем классе мы можем реализовать это следующим образом. А в коде создать экземпляр данного класса. И чтобы объекту, который мы только что создали, указать цвета, нам придется обратиться к свойствам данного объекта. Тут важно понять, как мне это сделать. важно понимать, что все переменные и функции внутри класса принято называть по-другому. В частности, переменные называются атрибутами или же свойствами класса, а функции называются методами. Хотя это простая терминология, но все же знать ее стоит. Ну и вместо того, чтобы указывать цвета отдельно через атрибуты объекта, мы можем сразу в классе создать новый метод, который будет называться init. И этот метод в Python принят в качестве зарезервированного для инициализации класса. И теперь мы можем указывать эти данные сразу при создании объекта. Именно так обычно и делают инициализацию каких-то данных объекта в реальном мире. Ну и сразу тут давайте попробуем применить наследование. Создадим еще один класс, назовем его KaleAlpha и унаследуем его от нашего класса Kale. В Python это делается вот таким простым образом. Только теперь дополним наш новый класс атрибутом alpha, который будет являться репрезентацией цветовой системы RGB с поддержкой альфа-канала. И тут же перепишем инициализатор, чтобы можно было указывать альфа-канал. При этом метод toHex мы не трогаем, потому что он у нас наследуется и менять его не нужно. Так как хекс-репрезентация цвета не поддерживает альфа-канал, а значит при переводе rgba в хекс атрибут alpha учитывать требуется. Ну и теперь мы можем создать еще и экземпляр класса KaleAlpha. Как видите, в этом нет ничего сложного. И при этом, если в будущем, например, обнаружится ошибка в функции перевода rgba в хекс, то мне нужно будет отредактировать только один кусок кода в классе Kale. Но это все я только что показал лишь в качестве какого-то практического примера, чтобы вы не остались с одной сухой теорией. А вообще в описании я также оставлю ссылку на годный урок по ООП конкретно в Пайтоне. Можете почитать и теперь, думаю, с новыми знаниями вы легко освоите объектно-ориентированное программирование. В остальном же я жду ваш царский лайк, надеюсь этот выпуск был для вас полезным. Конечно же подписывайтесь на канал и включайте колокольчик уведомлений, так мои выпуски хоть иногда, да будут появляться у вас на главной ютуба. Удачи и всегда помните, выучить можно что угодно. Было бы желание. Объяснение алгоритмов сортировки с примерами на Python В этой статье будут рассмотрены популярные алгоритмы, принципы их работы и реализация на Python. А ещё сравним, как быстро они сортируют элементы в списке. В качестве общего примера возьмём сортировку чисел в порядке возрастания. Но эти методы можно легко адаптировать под ваши потребности. Пузырьковая сортировка Этот простой алгоритм выполняет итерации по списку, сравнивая элементы попарно и меняя их местами, пока более крупные элементы не «всплывут» в начало списка, а более мелкие не останутся на «дне». Алгоритм Сначала сравниваются первые два элемента списка. Если первый элемент больше, они меняются местами. Если они уже в нужном порядке, оставляем их как есть. Затем переходим к следующей паре элементов, сравниваем их значения и меняем местами при необходимости. Этот процесс продолжается до последней пары элементов в списке. При достижении конца списка процесс повторяется заново для каждого элемента. Это крайне неэффективно, если в массиве нужно сделать, например, только один обмен. Алгоритм повторяется n² раз, даже если список уже отсортирован. Для оптимизации алгоритма нужно знать, когда его остановить, то есть когда список отсортирован. Чтобы остановить алгоритм по окончании сортировки, нужно ввести переменную-флаг. Когда значения меняются местами, устанавливаем флаг в значение True, чтобы повторить процесс сортировки. Если перестановок не произошло, флаг остаётся False и алгоритм останавливается. Алгоритм работает в цикле while и прерывается, когда элементы ни разу не меняются местами. Вначале присваиваем swapped значение True, чтобы алгоритм запустился хотя бы один раз. Время сортировки Если взять самый худший случай (изначально список отсортирован по убыванию), затраты времени будут равны O(n²), где n — количество элементов списка. Сортировка выборкой Этот алгоритм сегментирует список на две части: отсортированную и неотсортированную. Наименьший элемент удаляется из второго списка и добавляется в первый. Алгоритм На практике не нужно создавать новый список для отсортированных элементов. В качестве него используется крайняя левая часть списка. Находится наименьший элемент и меняется с первым местами. Теперь, когда нам известно, что первый элемент списка отсортирован, находим наименьший элемент из оставшихся и меняем местами со вторым. Повторяем это до тех пор, пока не останется последний элемент в списке. По мере увеличения значения i нужно проверять меньше элементов. Время сортировки Затраты времени на сортировку выборкой в среднем составляют O(n²), где n — количество элементов списка. Сортировка вставками Как и сортировка выборкой, этот алгоритм сегментирует список на две части: отсортированную и неотсортированную. Алгоритм перебирает второй сегмент и вставляет текущий элемент в правильную позицию первого сегмента. Алгоритм Предполагается, что первый элемент списка отсортирован. Переходим к следующему элементу, обозначим его х. Если х больше первого, оставляем его на своём месте. Если он меньше, копируем его на вторую позицию, а х устанавливаем как первый элемент. Переходя к другим элементам несортированного сегмента, перемещаем более крупные элементы в отсортированном сегменте вверх по списку, пока не встретим элемент меньше x или не дойдём до конца списка. В первом случае x помещается на правильную позицию. Время сортировки Время сортировки вставками в среднем равно O(n²), где n — количество элементов списка. Пирамидальная сортировка Также известна как сортировка кучей. Этот популярный алгоритм, как и сортировки вставками или выборкой, сегментирует список на две части: отсортированную и неотсортированную. Алгоритм преобразует второй сегмент списка в структуру данных «куча» (heap), чтобы можно было эффективно определить самый большой элемент. Алгоритм Сначала преобразуем список в Max Heap — бинарное дерево, где самый большой элемент является вершиной дерева. Затем помещаем этот элемент в конец списка. После перестраиваем Max Heap и снова помещаем новый наибольший элемент уже перед последним элементом в списке. Этот процесс построения кучи повторяется, пока все вершины дерева не будут удалены. Реализация Создадим вспомогательную функцию heapify() для реализации этого алгоритма: Время сортировки В среднем время сортировки кучей составляет O(n log n), что уже значительно быстрее предыдущих алгоритмов. Сортировка слиянием Этот алгоритм относится к алгоритмам «разделяй и властвуй». Он разбивает список на две части, каждую из них он разбивает ещё на две и т. д. Список разбивается пополам, пока не останутся единичные элементы. Соседние элементы становятся отсортированными парами. Затем эти пары объединяются и сортируются с другими парами. Этот процесс продолжается до тех пор, пока не отсортируются все элементы. Алгоритм Список рекурсивно разделяется пополам, пока в итоге не получатся списки размером в один элемент. Массив из одного элемента считается упорядоченным. Соседние элементы сравниваются и соединяются вместе. Это происходит до тех пор, пока не получится полный отсортированный список. Сортировка осуществляется путём сравнения наименьших элементов каждого подмассива. Первые элементы каждого подмассива сравниваются первыми. Наименьший элемент перемещается в результирующий массив. Счётчики результирующего массива и подмассива, откуда был взят элемент, увеличиваются на 1. Реализация Обратите внимание, что функция merge\_sort(), в отличие от предыдущих алгоритмов, возвращает новый список, а не сортирует существующий. Поэтому такая сортировка требует больше памяти для создания нового списка того же размера, что и входной список. Время сортировки В среднем время сортировки слиянием составляет O(n log n). Быстрая сортировка Этот алгоритм также относится к алгоритмам «разделяй и властвуй». Его используют чаще других алгоритмов, описанных в этой статье. При правильной конфигурации он чрезвычайно эффективен и не требует дополнительной памяти, в отличие от сортировки слиянием. Массив разделяется на две части по разные стороны от опорного элемента. В процессе сортировки элементы меньше опорного помещаются перед ним, а равные или большие —позади. Алгоритм Быстрая сортировка начинается с разбиения списка и выбора одного из элементов в качестве опорного. А всё остальное передвигаем так, чтобы этот элемент встал на своё место. Все элементы меньше него перемещаются влево, а равные и большие элементы перемещаются вправо. Реализация Существует много вариаций данного метода. Способ разбиения массива, рассмотренный здесь, соответствует схеме Хоара (создателя данного алгоритма). Время выполнения В среднем время выполнения быстрой сортировки составляет O(n log n). Обратите внимание, что алгоритм быстрой сортировки будет работать медленно, если опорный элемент равен наименьшему или наибольшему элементам списка. При таких условиях, в отличие от сортировок кучей и слиянием, обе из которых имеют в худшем случае время сортировки O(n log n), быстрая сортировка в худшем случае будет выполняться O(n²). Встроенные функции сортировки на Python Иногда полезно знать перечисленные выше алгоритмы, но в большинстве случаев разработчик, скорее всего, будет использовать функции сортировки, уже предоставленные в языке программирования. Отсортировать содержимое списка можно с помощью стандартного метода sort(): Или можно использовать функцию sorted() для создания нового отсортированного списка, оставив входной список нетронутым: Оба эти метода сортируют в порядке возрастания, но можно изменить порядок, установив для флага reverse значение True: В отличие от других алгоритмов, обе функции в Python могут сортировать также списки кортежей и классов. Функция sorted() может сортировать любую последовательность, которая включает списки, строки, кортежи, словари, наборы и пользовательские итераторы, которые вы можете создать. Функции в Python реализуют алгоритм Tim Sort, основанный на сортировке слиянием и сортировке вставкой. Сравнение скоростей сортировок Для сравнения сгенерируем массив из 5000 чисел от 0 до 1000. Затем определим время, необходимое для завершения каждого алгоритма. Повторим каждый метод 10 раз, чтобы можно было более точно установить, насколько каждый из них производителен. Пузырьковая сортировка — самый медленный из всех алгоритмов. Возможно, он будет полезен как введение в тему алгоритмов сортировки, но не подходит для практического использования. Быстрая сортировка хорошо оправдывает своё название, почти в два раза быстрее, чем сортировка слиянием, и не требуется дополнительное место для результирующего массива. Сортировка вставками выполняет меньше сравнений, чем сортировка выборкой и в реальности должна быть производительнее, но в данном эксперименте она выполняется немного медленней. Сортировка вставками делает гораздо больше обменов элементами. Если эти обмены занимают намного больше времени, чем сравнение самих элементов, то такой результат вполне закономерен. Вы познакомились с шестью различными алгоритмами сортировокна Python. Масштаб сравнения и количество перестановок, которые выполняет алгоритм вместе со средой выполнения кода, будут определяющими факторами в производительности. В реальных приложениях Python рекомендуется использовать встроенные функции сортировки, поскольку они реализованы именно для удобства разработчика.