[00:00.220 --> 00:09.980] Хаудихо, друзья! В комментариях все уже давно просят сделать ролик про ООП и объяснить, что же это такое и для чего оно вообще нужно.

[00:10.260 --> 00:15.800] Так что давайте познакомимся с, казалось бы, такой страшной штукой, как ООП.

[00:15.960 --> 00:26.900] 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.

[00:26.900 --> 00:34.940] И если вы вспомните, как начинали учиться программировать, то вас скорее всего учили писать код в процедурном стиле.

[00:35.060 --> 00:44.660] Это значит, что код, который вы писали, выполнялся последовательно и состоял преимущественно из всевозможных подпрограмм или же обычных функций.

[00:44.820 --> 00:49.820] Так вот, до изобретения ООП таким образом код писали по сути все.

[00:49.820 --> 00:56.880] И прикол тут в том, что пока ваша программа маленькая, процедурный стиль никак не мешает ее поддерживать.

[00:56.940 --> 00:58.520] Или как-то модифицировать.

[00:58.600 --> 01:05.880] То есть в коде условно всего 200-300 строчек кода, который легко читается и запутаться в нем по сути нельзя.

[01:06.260 --> 01:11.280] Но программисты очень быстро столкнулись с тем, что программы стали большими.

[01:11.420 --> 01:18.960] И вот когда в такой условной программе где-то 10 тысяч строк кода, то читать такой код становится уже невозможно.

[01:18.960 --> 01:25.900] И даже если сильно постараться все хорошо закомментировать и попробовать разбить этот код на файлы,

[01:26.040 --> 01:28.360] все равно останется масса проблем.

[01:28.580 --> 01:36.200] И вот для решения всей этой каши было придумано ООП, или же объектно ориентированное программирование.

[01:36.400 --> 01:41.760] Основная идея которого лежит в основе того, что все состоит из объектов.

[01:41.760 --> 01:45.480] В отличии, например, от того же процедурного программирования,

[01:45.640 --> 01:53.360] где каждая программа это один большой кусок кода, состоящий из множества переменных блоков и функций.

[01:53.580 --> 01:55.760] То есть любая объектно ориентированная программа,

[01:55.760 --> 02:01.220] программа это всегда набор компонентов взаимодействующих друг с другом и все они

[02:01.220 --> 02:08.420] называются объектами при этом здесь важно понимать то что программист создает объекты таким образом

[02:08.420 --> 02:14.660] чтобы они все участвовали в выполнении задачи но при этом каждый из них был бы обособленным

[02:14.660 --> 02:21.040] то есть каждый объект в программе отвечает только за себя и является полностью автономным

[02:21.040 --> 02:27.400] самостоятельным компонентом дальше каждый такой объект может быть представлением чего-то

[02:27.400 --> 02:33.980] реального например клиент магазина или товар которые могут являться частью целого интернет

[02:33.980 --> 02:40.420] магазина либо объект может представлять что-то абстрактное что существует только в компьютере

[02:40.420 --> 02:47.540] условно это может быть какая-то структура данных . на экране или что-то в таком духе но для программы

[02:47.540 --> 02:51.020] это все не более чем просто набор объектов каждый объект может быть объектом который

[02:51.020 --> 02:51.200] в этом доме или что-то в таком духе но при этом каждый объект может быть объектом который

[02:51.200 --> 02:57.580] из которых общается с программой и взаимодействует с другими объектами посредством своего публичного

[02:57.580 --> 03:04.980] интерфейса а вот вся его внутренняя реализация она полностью скрыта и такой механизм называется

[03:04.980 --> 03:12.380] инкапсуляции то есть реализация объекта и его данные скрываются от других компонентов это

[03:12.380 --> 03:18.000] бывает полезно тогда когда к примеру доступ к какой-то переменной нужно предоставлять не

[03:18.000 --> 03:21.000] напрямую а с помощью специальных методов для выполнения и довольно-таки специальных методов для

[03:21.000 --> 03:26.660] чтения и записи. Да и вообще, если хорошо продумать и спроектировать всю программу,

[03:26.740 --> 03:32.060] то благодаря инкапсуляции можно будет работать над каждой ее частью отдельно,

[03:32.160 --> 03:38.780] не охватывая умом всю программу целиком. И еще в ООП есть такая штука, благодаря которой

[03:38.780 --> 03:45.400] можно легко создавать огромные объекты, разбивая их на несколько маленьких. Другими словами,

[03:45.400 --> 03:50.620] такие объекты могут собираться сразу из нескольких других, выступая своего рода

[03:50.620 --> 03:57.320] агрегатором реализации. Например, можно взять два объекта, компьютер и монитор. Каждый из них

[03:57.320 --> 04:03.700] реализует какой-то свой функционал. И дальше можно создать большой объект лэптоп, состоящий

[04:03.700 --> 04:09.540] из более маленьких объектов компьютер и монитор. И такой принцип дает возможность сильно

[04:09.540 --> 04:15.300] упростить сложность всей программы, а называется он наследование. Это еще один

[04:15.300 --> 04:21.520] мощный инструмент объектно-ориентированного программирования. Все это позволяет создавать

[04:21.520 --> 04:28.420] очень большие программы, над которыми могут работать сразу сотни или десятки тысяч программистов.

[04:28.420 --> 04:35.280] Но при этом все они могут не знать, как устроена вся программа целиком, потому что каждый из них

[04:35.280 --> 04:42.440] будет работать лишь над ее отдельными частями. Ну и третий мощный инструмент ООП, который пока

[04:42.440 --> 04:45.280] что можно пропустить, это полиморфизм.

[04:45.300 --> 04:51.800] Потому что понимается он гораздо лучше тогда, когда вы уже изучите концепции ООП на практике.

[04:51.800 --> 04:59.280] Пока что можете запомнить лишь то, что полиморфизм это когда функция способна обработать данные разных

[04:59.280 --> 05:06.920] типов. Ну и по сути это все инструменты, которые имеются в ООП. Их всего три, а именно инкапсуляция,

[05:06.920 --> 05:13.620] наследование и полиморфизм. И как видите, ничего сложного в этом нет. Соответственно, сегодня

[05:13.620 --> 05:15.280] практически все языки в ООП имеют полиморфизм. И если вы хотите узнать больше о полиморфизме,

[05:15.280 --> 05:16.280] то смотрите это видео до конца. В этом видео я расскажу о том, что полиморфизм и полиморфизм

[05:16.280 --> 05:24.460] являются объектно-ориентированными. Сюда относятся и Python, и Java, и PHP, и C++, и многие другие.

[05:24.460 --> 05:30.540] Так что понимая общий принцип работы ООП, вы заложите в себе фундамент знаний любого

[05:30.540 --> 05:37.000] объектно-ориентированного языка программирования. И особенно это касается языка Python, потому что

[05:37.000 --> 05:42.280] хоть он и допускает различные методологии программирования, в его основе лежит все-таки

[05:42.280 --> 05:45.260] ООП. Поэтому абсолютно все в Python. Если вы хотите узнать больше о полиморфизме и полиморфизме,

[05:45.280 --> 05:50.140] то смотрите это видео до конца. И если вы хотите узнать больше о полиморфизме, то смотрите это видео до конца.

[05:50.140 --> 05:56.440] Ну а если вы еще только начинаете изучать Python или планируете это сделать, то порекомендую

[05:56.440 --> 06:03.360] обратить внимание на курс Python для веб-разработки. Занятия здесь будут длиться 9 месяцев, и это как

[06:03.360 --> 06:09.200] раз то время, за которое вас нехило натаскают и вы действительно научитесь программировать. В том

[06:09.200 --> 06:15.120] числе большое количество времени учебы уделяется объектно-ориентированному программированию. А

[06:15.120 --> 06:15.260] походу, это не только для вас, а для всех, кто не знает, что это. А это очень важный момент,

[06:15.260 --> 06:22.580] ходу учебы выдаются домашние задания и создается настоящий проект. Ну а для записи на курс вам не

[06:22.580 --> 06:28.640] нужны никакие стартовые знания, всему научат на месте. Все, что от вас требуется, это желание

[06:28.640 --> 06:35.460] стать Python-разработчиком. В общем, все, что нужно для качественного старта. Ну а ссылку на курс я

[06:35.460 --> 06:44.320] оставлю в описании под роликом. Сообщите менеджеру промокод Howdyho и получите 35% скидку до 20 мая.

[06:45.260 --> 06:52.060] Окей, сейчас вам может показаться, что вы хорошо поняли, что такое ООП и знаете его основные

[06:52.060 --> 06:58.180] преимущества. И это, конечно, хорошо, но пока еще рано переходить к практике. До этого вы

[06:58.180 --> 07:05.260] обязательно должны понять, что такое класс и объект. И это действительно важно. Настолько важно,

[07:05.440 --> 07:11.780] что я даже скажу такую штуку. Если вы четко поймете, что такое класс и объект, то можете смело

[07:11.780 --> 07:15.220] считать, что вы знаете как минимум половину всего ООП.

[07:15.260 --> 07:22.680] И я сейчас не шучу. Так вот, объяснить, что такое класс, можно в принципе по-разному. Но если делать

[07:22.680 --> 07:29.820] это совсем по-простому, то класс это просто кусок кода, у которого есть какое-то имя. И чтобы

[07:29.820 --> 07:37.280] воспользоваться этим куском кода, нужно из этого класса создать объект. Эту процедуру еще называют

[07:37.280 --> 07:44.340] инициализацией класса. Или созданием экземпляра класса. Ну или просто созданием объекта. Как не

[07:45.260 --> 07:52.300] все одно и то же. И такой объект принято называть экземпляром класса. Но не будем пока забегать

[07:52.300 --> 07:58.760] так далеко. Чтобы вы лучше понимали, давайте представим себе какой-то объект из реальной жизни.

[07:59.020 --> 08:05.200] Ведь, как вы помните, концепция ООП подразумевает, что все является объектами. Соответственно,

[08:05.200 --> 08:12.820] вокруг нас очень много разных объектов. Это стол, стул, дом, собака, кошка, другие люди, ручка,

[08:15.260 --> 08:23.020] и так далее. Для примера давайте возьмем автомобиль. И подумаем, что нужно, чтобы создать автомобиль.

[08:23.020 --> 08:31.020] В первую очередь это, конечно же, чертеж, в котором будет описано, как и из чего собирается авто. Ну и

[08:31.020 --> 08:37.580] там же будет описан полный список нужных деталей. Так вот, если для того, чтобы собрать машину,

[08:37.580 --> 08:45.020] нам нужен чертеж, то в программировании для того, чтобы создать объект, нам нужен класс. И думаю, вы уже начинаете понимать, как это работает.

[08:45.260 --> 08:56.140] Соответственно, класс — это тоже своего рода чертеж, в котором довольно точно перечислено, из чего он состоит и какой реализует функционал.

[08:56.140 --> 09:01.740] Ну и так как в жизни мы можем из одного чертежа создать сколько угодно автомобилей,

[09:01.740 --> 09:09.780] единственный наш лимит — это количество оставшихся деталей или же ресурсов, в программировании это устроено похожим образом.

[09:09.780 --> 09:14.060] Из одного класса можно создать сколько угодно объектов.

[09:14.060 --> 09:15.020] А ресурсом будет выставка.

[09:15.020 --> 09:17.540] И в этом случае будет выступать доступная мощность компьютера.

[09:17.540 --> 09:22.420] То есть, чем больше мы захотим создать объектов, тем больше потребуется памяти.

[09:22.420 --> 09:29.140] И для примера давайте на Python создадим какой-то простенький класс в качестве небольшой практики.

[09:29.140 --> 09:36.260] Пусть этот класс будет называться color, и в нем будет ровно три переменные, а именно red, green и blue.

[09:36.260 --> 09:42.380] И как вы уже понимаете, мы создали класс для репрезентации цветовой системы RGB.

[09:42.380 --> 09:44.780] Это уже не какие-то аналогии, а вполне возможности.

[09:44.780 --> 09:52.180] И это вполне реальный класс, который особенно часто встречается в игровых движках или библиотеках для работы с графикой.

[09:52.180 --> 09:58.940] В таких классах обычно существует метод для перевода RGB в HEX-репрезентацию цвета,

[09:58.940 --> 10:03.180] и в нашем классе мы можем реализовать это следующим образом.

[10:03.180 --> 10:06.780] А в коде создать экземпляр данного класса.

[10:06.780 --> 10:13.980] И чтобы объекту, который мы только что создали, указать цвета, нам придется обратиться к свойствам данного объекта.

[10:13.980 --> 10:14.180] Тут важно понять, как мне это сделать.

[10:14.180 --> 10:20.700] важно понимать, что все переменные и функции внутри класса принято называть по-другому. В частности,

[10:20.840 --> 10:27.480] переменные называются атрибутами или же свойствами класса, а функции называются методами. Хотя

[10:27.480 --> 10:34.680] это простая терминология, но все же знать ее стоит. Ну и вместо того, чтобы указывать цвета отдельно

[10:34.680 --> 10:40.560] через атрибуты объекта, мы можем сразу в классе создать новый метод, который будет называться

[10:40.560 --> 10:47.180] init. И этот метод в Python принят в качестве зарезервированного для инициализации класса. И

[10:47.180 --> 10:53.240] теперь мы можем указывать эти данные сразу при создании объекта. Именно так обычно и делают

[10:53.240 --> 10:59.460] инициализацию каких-то данных объекта в реальном мире. Ну и сразу тут давайте попробуем применить

[10:59.460 --> 11:07.280] наследование. Создадим еще один класс, назовем его KaleAlpha и унаследуем его от нашего класса

[11:07.280 --> 11:10.520] Kale. В Python это делается вот таким простым образом.

[11:10.560 --> 11:15.720] Только теперь дополним наш новый класс атрибутом alpha, который будет являться

[11:15.720 --> 11:22.580] репрезентацией цветовой системы RGB с поддержкой альфа-канала. И тут же перепишем инициализатор,

[11:22.580 --> 11:27.720] чтобы можно было указывать альфа-канал. При этом метод toHex мы не трогаем,

[11:27.720 --> 11:33.900] потому что он у нас наследуется и менять его не нужно. Так как хекс-репрезентация цвета

[11:33.900 --> 11:40.180] не поддерживает альфа-канал, а значит при переводе rgba в хекс атрибут alpha учитывать

[11:40.560 --> 11:46.740] требуется. Ну и теперь мы можем создать еще и экземпляр класса KaleAlpha. Как видите,

[11:46.740 --> 11:52.780] в этом нет ничего сложного. И при этом, если в будущем, например, обнаружится ошибка в функции

[11:52.780 --> 11:58.080] перевода rgba в хекс, то мне нужно будет отредактировать только один кусок кода в

[11:58.080 --> 12:04.620] классе Kale. Но это все я только что показал лишь в качестве какого-то практического примера,

[12:04.620 --> 12:10.400] чтобы вы не остались с одной сухой теорией. А вообще в описании я также оставлю ссылку

[12:10.400 --> 12:17.660] на годный урок по ООП конкретно в Пайтоне. Можете почитать и теперь, думаю, с новыми знаниями вы

[12:17.660 --> 12:23.900] легко освоите объектно-ориентированное программирование. В остальном же я жду ваш

[12:23.900 --> 12:29.640] царский лайк, надеюсь этот выпуск был для вас полезным. Конечно же подписывайтесь на канал и

[12:29.640 --> 12:36.020] включайте колокольчик уведомлений, так мои выпуски хоть иногда, да будут появляться у вас на главной

[12:36.020 --> 12:40.380] ютуба. Удачи и всегда помните, выучить можно что угодно.

[12:40.400 --> 12:41.780] Было бы желание.