# 5 Наследование

Наследование – важная составляющая объектно-ориентированного программирования. Так или иначе мы уже сталкивались с ним, ведь объекты наследуют атрибуты своих классов. Однако обычно под наследованием в ООП понимается наличие классов и подклассов. Также их называют супер- или надклассами и классами, а также родительскими и дочерними классами.

Суть наследования здесь схожа с наследованием объектами от классов. Дочерние классы наследуют атрибуты родительских, а также могут переопределять атрибуты и добавлять свои.

# 6 Инкапсуляция

Инкапсуляция — ограничение доступа к составляющим объект компонентам (методам и переменным). Инкапсуляция делает некоторые из компонент доступными только внутри класса.

Инкапсуляция в Python работает лишь на уровне соглашения между программистами о том, какие атрибуты являются общедоступными, а какие — внутренними.

Одиночное подчеркивание в начале имени атрибута говорит о том, что переменная или метод не предназначен для использования вне методов класса, однако атрибут доступен по этому имени.

Какие модификаторы доступа Вы знаете?

private и protected.

# 7 Полиморфизм

Полиморфизм - разное поведение одного и того же метода в разных классах. Например, мы можем сложить два числа, и можем сложить две строки. При этом получим разный результат, так как числа и строки являются разными классами.

# 8 Дайте определение термину "абстракция"

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

# 9 Что такое композиция?

**Композиция** — это концепция, которая моделирует отношения. Она позволяет создавать сложные типы, комбинируя объекты других типов. Это означает, что класс **Composite** может содержать объект другого класса **Component**.

 В чем основное отличие "композиции" от "наследования"?

**Наследование** и **композиция** являются двумя основными понятиями в объектно-ориентированном программировании, которые моделируют отношения между двумя классами. Они определяют дизайн приложения и определяют, как приложение должно развиваться по мере добавления новых функций или изменения требований.

Оба они реализуют повторное использование кода, но делают это по-разному.

# 10 Дайте определение термину "итератор"?

Итератор (iterator) - это объект, который возвращает свои элементы по одному за раз.

С точки зрения Python - это любой объект, у которого есть метод \_\_next\_\_. Этот метод возвращает следующий элемент, если он есть, или возвращает исключение StopIteration, когда элементы закончились.

Кроме того, итератор запоминает, на каком объекте он остановился в последнюю итерацию.

В Python у каждого итератора присутствует метод \_\_iter\_\_ - то есть, любой итератор является итерируемым объектом. Этот метод просто возвращает сам итератор

11 Что такое "генератор"?

Генераторы — это функции, которые можно приостанавливать и возобновлять во время их выполнения, при этом они возвращают объект, который можно итерировать. В отличие от списков, они [ленивы](https://en.wikipedia.org/wiki/Lazy_evaluation) и поэтому работают с текущим элемент только по запросу. Таким образом, они намного эффективнее используют память при работе с большими наборами данных. Генераторы позволяют нам запрашивать значения по мере необходимости, делая наши приложения более эффективными в использовании памяти и идеально подходящими для бесконечных потоков данных. Они также могут быть использованы для рефакторинга обработки из циклов, что приводит к более чистому, разъединенному коду.

**В чем его главное отличие от классических итераторов?**

Каждый генератор является итератором, но не наоборот. Генератор создается путем вызова функции, которая имеет одно или несколько yieldвыражений ( yieldоператоров в Python 2.5 и более ранних версиях) и является объектом, который соответствует определению предыдущего абзаца iterator.

Вы можете захотеть использовать пользовательский итератор, а не генератор, когда вам нужен класс с несколько сложным поведением, поддерживающим состояние, или вы хотите представить другие методы помимо next(и \_\_iter\_\_и \_\_init\_\_). Чаще всего достаточно генератора (иногда, для достаточно простых нужд, выражения генератора ), и его проще кодировать, потому что поддержание состояния (в разумных пределах) в основном «выполняется для вас», когда кадр приостанавливается и возобновляется.

# 12 Что такое "метаклассы"?

Метаклассы – это такие классы, экземпляры которых сами являются классами. Подобно тому, как «обычный» класс определяет поведение экземпляров класса, метакласс определяет и поведение классов, и поведение их экземпляров.

Метаклассы поддерживаются не всеми объектно-ориентированными языками программирования. Те языки программирования, которые их поддерживают, значительно отличаются по способу их реализации. Но в Python метаклассы есть.

Некоторые программисты рассматривают метаклассы в Python как «решения, которые ждут или ищут задачу».

У метаклассов множество применений. Выделим несколько из них:

* Логирование и профилирование;
* Тестирование интерфейса;
* Регистрация классов во время создания;
* Автоматическое создание свойств;
* Прокси;
* Автоматическая блокировка/синхронизация ресурсов.

### **Определение метаклассов**

В целом, метаклассы определяются также, как и любые другие классы в Python, но это классы, которые наследуются от «типа». Еще одно отличие в том, что метакласс вызывается автоматически, когда оператор класса, использующий метакласс, заканчивается. Другими словами, если ключевое слово metaclass не передается после базовых классов заголовка класса (однако базовых классов может и не быть), то будет вызван type() (т.е. \_call\_ типа). С другой стороны, если ключевое слово metaclass используется, то назначенный ему класс будет вызываться вместо type.

**Что такое "метапрограммирование"?**

**Метапрограммирование** — вид [программирования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), связанный с созданием [программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), которые порождают другие программы как результат своей работы (в частности, на стадии [компиляции](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) их [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)), либо программ, которые меняют себя во время выполнения [самомодифицирующийся код](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%BC%D0%BE%D0%BC%D0%BE%D0%B4%D0%B8%D1%84%D0%B8%D1%86%D0%B8%D1%80%D1%83%D1%8E%D1%89%D0%B8%D0%B9%D1%81%D1%8F_%D0%BA%D0%BE%D0%B4) Первое позволяет получать программы при меньших затратах времени и усилий на кодирование, чем если бы программист писал их вручную целиком, второе позволяет улучшить свойства кода (размер и быстродействие).

# 13 Что такое "миксины"?

В Python так называемые **миксины — это классы, которые живут в обычном дереве наследования**, но они остаются небольшими, чтобы избежать создания иерархий, которые слишком сложны для понимания программистом. В частности, миксины не должны иметь общих предков, кроме объекта, с другими родительскими классами.

## Для чего они нужны?

Смешанные классы используются для добавления простых изменений в классы.

Миксины реализованы в Python с использованием множественного наследования: они обладают большой выразительной силой, но требуют тщательного проектирования.

# 14 Как создать экземпляр класса (объект)?

### Создание экземпляра

Для создания [объекта](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) — экземпляра класса (то есть, инстанцирования класса), достаточно вызвать класс по имени и задать параметры конструктора.

Переопределив классовый метод \_\_new\_\_, можно управлять процессом создания экземпляра. Этот метод вызывается до метода \_\_init\_\_ и должен вернуть новый экземпляр либо None (в последнем случае будет вызван \_\_new\_\_ родительского класса). Метод \_\_new\_\_ используется для управления созданием неизменчивых (immutable) объектов, управления созданием объектов в случаях, когда \_\_init\_\_ не вызывается, например, при десериализации (unpickle).