Классы в языке Python

Объектно-ориентированное программирование в Python

Классы предоставляют средства объединения данных и функциональности вместе. Создание нового класса создает новый тип объекта, позволяя создавать новые экземпляры этого типа. К каждому экземпляру класса могут быть прикреплены атрибуты для поддержания его состояния. Экземпляры класса также могут иметь методы, определяемые его классом, для изменения его состояния.

По сравнению с другими языками программирования, механизм классов Python добавляет классы с минимумом нового синтаксиса и семантики. Это смесь механизмов класса, взятых из языков программирования C++ и Modula-3. Классы Python предоставляют все стандартные возможности объектно-ориентированного программирования:

- механизм наследования классов позволяет использовать несколько базовых классов,
- производный класс может переопределять любые методы своего базового класса или классов,
- метод может вызывать метод базового класса с тем же именем.
- объекты могут содержать произвольные объемы и типы данных.
- как и в случае с модулями, классы имеют динамическую природу Python: они создаются во время выполнения и могут быть изменены после создания.

В терминологии языка C++, как правило, члены класса, включая элементы данных, являются открытыми (за исключением закрытых переменных) и все функции-члены являются виртуальными. В Python нет сокращений для ссылки на члены объекта из его методов: функция метода объявляется с явным первым аргументом `self`, представляющим объект, который неявно предоставляется вызовом. Классы в Python сами по себе являются объектами, что обеспечивает семантику для импорта и переименования. В отличие от C++ и Modula-3, встроенные типы могут использоваться в качестве базовых классов для расширения их пользователем. Также, как и в C++, большинство встроенных операторов со специальным синтаксисом (арифметические операторы, подписка и т. д.) можно переопределить для экземпляров класса.

Объектно-ориентированная семантика Modula-3 ближе к семантике Python, чем C++.

Имена и объекты классов.

Объекты имеют индивидуальность, и несколько имен (в нескольких областях видимости) могут быть связаны с одним и тем же объектом. В других языках это называется псевдонимами. Это обычно не принимается во внимание при первом взгляде на Pyt и может быть безопасно проигнорировано при работе с неизменяемыми базовыми типами - числами, строками, кортежами. Однако псевдонимы могут иметь неожиданный эффект на семантику кода Python, использующего изменяемые объекты, такие как списки, словари и большинство других типов.

Такое поведение используется в интересах программы, так как псевдонимы ведут себя как указатели в некоторых отношениях. Например, передача объекта является дешевой, так как реализация передает только указатель и если функция изменяет объект, передаваемый в качестве аргумента, вызывающий объект увидит изменение - это устраняет необходимость в двух различных механизмах передачи аргументов, как в языке Pascal.

Пространство имен и область видимости в классах

Пространство имен - это отображение имен, определенных в объектах. Большинство пространств имен в настоящее время реализованы в виде словарей Python. Области видимости определяются статически, но используются динамически. В любой момент во время выпо

Определение классов в Python

Определения классов, так-же как определения функций, должны быть выполнены до того, как они окажут какой-либо эффект. Вполне возможно, что определение класса можно поместить в ветвь оператора if или определить класс внутри функции.

Объект класса и конструктор класса в Python

Объекты класса поддерживают два вида операций: ссылки на атрибуты и создание экземпляров. Атрибуты - это имена, которые были определены в пространстве имен класса при создании объекта класса. Операция создания экземпляра - ”вызов" объекта класса.

Создание экземпляра класса в Python

Единственные операции, понятные экземплярам класса, являются ссылками на атрибуты. Существует два вида допустимых имен атрибутов, атрибуты данных и методы класса.

Метод экземпляра класса в Python

Объект метода создается путем упаковки указателей объекта экземпляра и объекта функции, только что найденных вместе в абстрактном объекте: это объект метода. Когда объект метода вызывается со списком аргументов, новый список аргументов создается из э

Что такое метод класса в Python и зачем нужен

Обычно в Python методы класса отмечаются декоратором @classmethod. Метод класса, вместо того чтобы принимать аргумент self, принимает аргумент cls. При вызове метода этот аргумент указывает на сам класс, а не на экземпляр класса. Следовательно он не может изменять состояние экземпляра объекта.

Что такое статический метод в классах Python и зачем нужен

Статические методы класса отмечаются декоратором @staticmethod. Этот тип метода не принимает ни параметра self как метод экземпляра класса, ни параметра cls как метод класса. При этом, конечно, статический метод может принимать произвольное количество других параметров. Не может изменять ни состоян

Атрибуты класса и переменные экземпляра класса в Python

Переменные экземпляра класса предназначены для данных, уникальных для каждого экземпляра класса, а переменные класса/ атрибуты данных класса - для атрибутов и методов, общих для всех экземпляров класса.

Кэширование методов экземпляра декоратором lru_cache в Python

При использовании декоратора functools.lru_cache для кэширования метода, экземпляры класса, инкапсулирующего этот метод, никогда не будут собираться сборщиком мусора Python в течение всего времени существования удерживающего их процесса.

Закрытые/приватные методы и переменные класса Python

Имя с префиксом подчеркивания, например _spam, следует рассматривать как непубличную часть API, будь то функция, метод или элемент данных. Это следует считать деталями реализации и могут быть изменены без предварительного уведомления.

Наследование классов в Python

Выполнение определения производного класса `DerivedClassName` происходит так же, как и для базового класса `BaseClassName`. Когда объект класса создан, базовый класс `BaseClassName` запоминается. Это используется для разрешения ссылок на атрибуты.

Множественное наследование классов в Python

Порядок разрешения метода (MRO - method resolution order) динамически изменяется для поддержки совместных вызовов super().

Абстрактные классы в Python

Абстрактный класс определяет общий интерфейс для набора подклассов. Он предоставляет общие атрибуты и методы для всех подклассов, чтобы уменьшить дублирование кода. Он также заставляет подклассы реализовывать абстрактные методы, чтобы избежать каких-то несоответствий.

Перегрузка методов в классе Python

Перегрузка методов класса (универсальные методы) - это распространенный шаблон программирования (множественная диспетчеризация/отправка) применяемый в статических типизированных языках. В динамически типизированных языках перегрузка методов невозможна, тем не менее, в Python есть простой способ реа

Что такое миксины в Python и как их использовать

Миксины или как еще их называют примеси - это ограниченная форма множественного наследования. В частности, в контексте языка Python, миксин - это родительский класс, который предоставляет функциональные возможности подклассам, но не предназначен для создания экземпляров самого себя.

Класс Python как структура данных, подобная языку C

Куску кода Python, который ожидает определенный абстрактный тип данных, часто можно передать класс, который эмулирует методы этого типа данных.

Создание пользовательских типов данных в Python

Python позволяет программистам, с помощью обычных классов, создавать свои собственные типы данных - типы, которые группируют несколько отдельных переменных вместе. Такие типы позволяет группировать переменные разных типов в единое целое.

Специальные (магические) методы класса Python

Класс может реализовывать определенные операции, которые вызываются специальным синтаксисом (например, арифметические операции или индексирование и срезы), определяя методы со специальными именами. Это подход Python к перегрузке операторов, позволяющий классам определять собственное поведение по от

Базовая настройка классов Python магическими методами

В разделе рассмотрена базовая пользовательская настройка классов Python специальными (магическими) методами. Предназначение методов: __new__, __init__, , __del__, __repr__, __str__, __bytes__, __format__, __hash__, __bool__, а та[Вверх]е методы сравнения.

Настройка доступа к атрибутам класса Python

В разделе рассматриваются специальные (магические) методы, которые могут быть определены для настройки поведения при доступе к атрибутам экземпляров класса: __getattr__, __getattribute__, __setattr__, __deltattr__, __dir__.

Дескриптор класса для чайников в Python

Этот материал дает базовый обзор дескриптора класса, плавно переходя от простых примеров, добавляя по одной функции за раз. Начните здесь, если вы новичок в дескрипторах.

Протокол дескриптора класса в Python

В общем случае, дескриптор - это атрибут объекта с "привязкой поведения", доступ к которому был переопределен методами в протоколе дескриптора. Этими методами являются __get__(), __set__() и __delete__(). Если какой-либо из этих методов определен для объекта, то он называется дескриптором.

Практический пример дескриптора в Python

В этом материале создается практичный и мощный инструмент (валидатор) для обнаружения известных ошибок, связанных с повреждением данных.

Использование метода .__new__() в классах Python

Обычно нет необходимости создавать собственную реализацию специального метода .__new__(), но есть несколько вариантов использования: создание подклассов неизменяемых типов, быстрое создание паттерна Singleton или возможность возвращать экземпляры другого класса.

Специальный атрибут __slots__ класса Python

Специальный (магический) метод __slots__ позволяют явно объявлять элементы данных (например, свойства) и запрещать создание словаря __dict__ и __weakref__. Пространство, сэкономленное от НЕ использования __dict__, может быть значительным! Скорость поиска атрибутов также может быть значительно увели

Специальный метод __init_subclass__ класса Python

Когда класс наследуется от другого класса, то для этого класса вызывается метод __init_subclass__. Таким образом, можно писать классы, которые изменяют поведение подклассов.

Определение метаклассов metaclass в Python

По умолчанию классы создаются с использованием класса type(). Тело класса выполняется в новом пространстве имен, а имя класса локально привязано к результату выполнения type(name, bases, namespace). Процесс создания класса можно настроить, передав ключевой аргумент metaclass в строке определения кл

Эмуляция контейнерных типов в классах Python

Следующие методы могут быть определены для реализации контейнерных объектов. Рассмотрены магические методы: __len__, __length_hint__, __getitem__, __setitem__, __delitem__, __missing__, __reversed__ и __contains__.

Другие специальные методы класса в Python

Настройка проверок экземпляра и подкласса. Эмуляция универсальных "generic" типов. Эмуляция вызываемых объектов.

Как Python ищет специальные методы в классах

Для пользовательских классов неявные вызовы специальных методов гарантированно работают правильно только в том случае, если они определены для типа объекта (в теле класса как фактический метод), а не в словаре экземпляра объекта.

Шаблон проектирования Фабрика и его реализация в Python

Шаблон Factory используется для создания конкретных реализаций общего интерфейса. Он отделяет процесс создания объекта от кода, который зависит от интерфейса объекта. То есть приложение делегирует это решение отдельному компоненту, который создает конкретный объект.

Содержание раздела:

Вверх

Вверх