

Python Essential

Наследование и полиморфизм

Python Essential

После урока обязательно



Повторите этот урок в видео формате на [ITVDN.com](http://itvdn.com)

Доступ можно получить через руководство вашего учебного центра

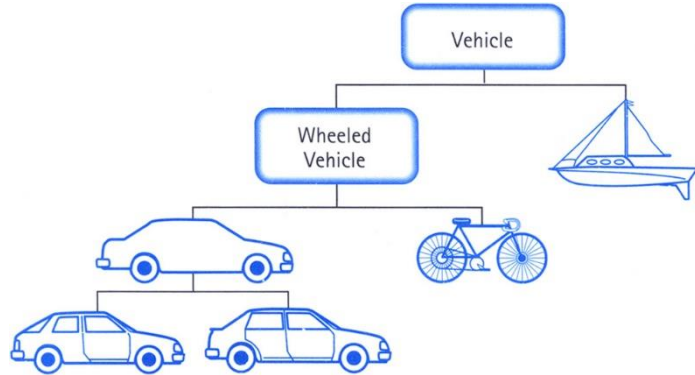


Проверьте как Вы усвоили данный материал на [TestProvider.com](http://testprovider.com)

Наследование и полиморфизм

Наследование и полиморфизм

Наследование



- **Наследование** — механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса.
- Класс-потомок может добавить собственные методы и свойства, а также пользоваться родительскими методами и свойствами.
- Позволяет строить иерархии классов.
- Является одним из основных принципов объектно-ориентированного программирования.

Наследование и полиморфизм

Классы старого и нового типа



- В версиях до 2.2 некоторые объектно-ориентированные возможности Python были заметно ограничены. Начиная с версии 2.2, объектная система Python была существенно переработана и дополнена.
- В целях совместимости с существующим кодом в Python 2 существуют две системы типов: *классы нового типа* (new-style classes) и *классы старого типа* (old-style classes, classic classes).
- Для создания класса нового типа следует унаследовать его от любого другого класса нового типа. Все стандартные классы являются классами нового типа. Базовым из них является класс **object**.
- Если в Python 2 не указывать базовый класс или унаследовать его от другого класса старого типа, то данный класс является классом старого типа.



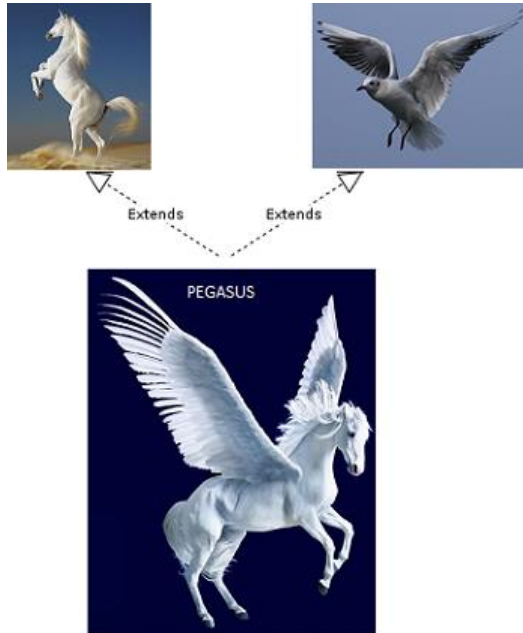
**Классы старого типа нужны только для обратной совместимости.
В новом коде следует использовать только классы нового типа.**



В Python 3 все классы являются классами нового типа и наследуются по умолчанию от *object*.

Наследование и полиморфизм

Множественное наследование



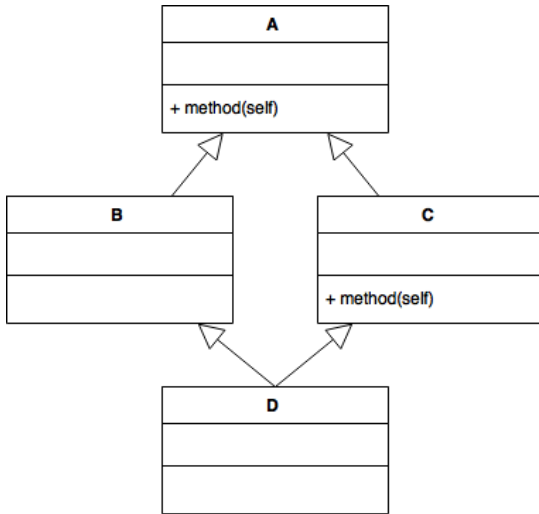
При **множественном наследовании** у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинство такого подхода в большей гибкости, однако он может быть потенциальным источником ошибок.

Список базовых классов указывается через запятую в круглых скобках после имени данного класса:

```
class Pegasus(Horse, Bird):  
    pass
```

Наследование и полиморфизм

Порядок разрешения методов (MRO)



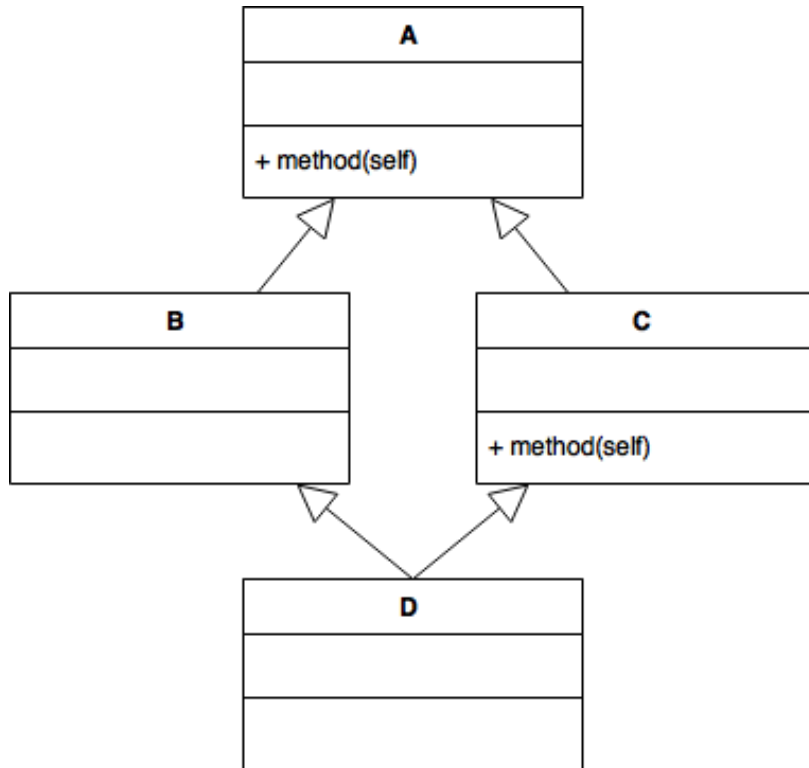
- Если атрибут, к которому осуществляется доступ, не найден в текущем классе, производится его поиск в классах-родителях.
- Порядок, в котором интерпретатор просматривает базовые классы, определяется **линеаризацией** данного класса, также называемой **MRO (Method Resolution Order)**. Она хранится в атрибуте класса `__mro__`.
- MRO строится при помощи алгоритма C3-линеаризации.
- Свойства линеаризации:
 - *устойчивость* и расширяемость;
 - *монотонность*: в линеаризации класса-потомка соблюдается тот же порядок следования классов-прородителей, что и в линеаризации класса-родителя;
 - *свойство локального старшинства*: в линеаризации класса-потомка соблюдается тот же порядок следования классов-родителей, что и в его объявлении



Линеаризация строится только для классов нового типа. В классах старого типа поиск атрибутов производится при помощи поиска в глубину, что может давать некорректные результаты при ромбовидном наследовании.

Наследование и полиморфизм

Порядок разрешения методов (MRO)



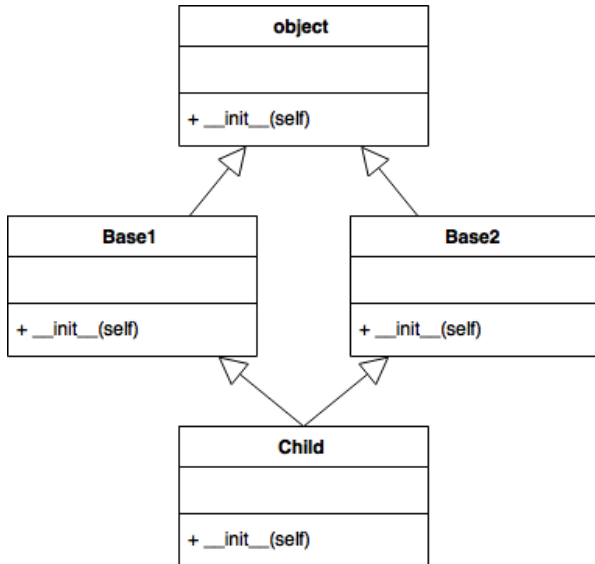
В примере слева линейаризация класса D, то есть `D.__mro__`, выглядит как `[D, B, C, A]`.

Таким образом, если попытаться вызвать `obj.method()`, где `obj` – экземпляр класса D, будет вызван метод класса C.

Если бы A был классом старого типа в Python 2, то был бы вызван метод `method` класса A, что в данном случае является неправильным поведением, так как он переопределён в классе C, наследником которого является класс D. Это одна из причин, по которым следует использовать классы нового типа.

Наследование и полиморфизм

Получение доступа к атрибутам суперкласса



- Если в данном классе метод или атрибут был переопределён, а требуется доступ к соответствующему атрибуту суперкласса, это можно совершить двумя способами:
 - путём явного обращения к атрибуту необходимого класса:
`BaseClass.method(self)`
 - при помощи инстанцирования специального прокси-объекта класса `super` (выглядит, как вызов функции).
- В Python 2 как параметры конструктора `super` передаются имя текущего класса и ссылка на экземпляр текущего класса:
`super(MyClass, self).method()`
- В Python 3 можно не указывать ничего и данные параметры будут получены автоматически:
`super().method()` # то же самое, что `super(__class__, self).method()`



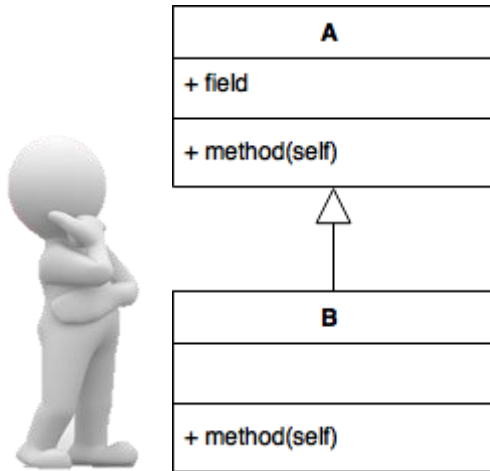
super возвращает специальный промежуточный объект, который предоставляет доступ к атрибутам следующего класса в `__mro__`.



super недоступен для классов старого типа

Наследование и полиморфизм

Определение типа объекта



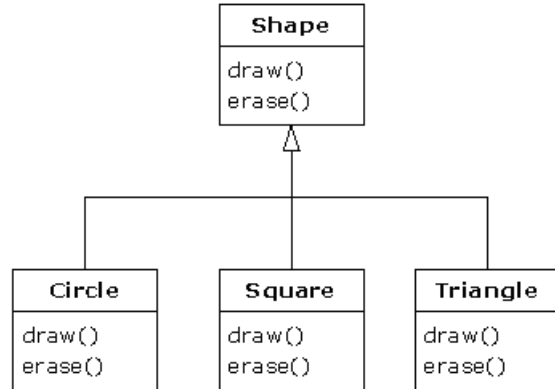
- Тип* данного объекта можно определить при помощи атрибута `__class__` и встроенной функции `type(obj)`.
- Атрибут класса `__bases__` хранит кортеж (неизменяемый список) базовых классов.
- Поскольку отношение наследования является транзитивным, в общем случае для проверки того, является ли данный объект экземпляром заданного класса или является ли данный класс подклассом заданного класса, эти атрибуты нужно проверять рекурсивно. Существуют встроенные функции, которые это делают.
- `isinstance(obj, cls)` проверяет, является ли *obj* экземпляром класса *cls* или класса, который является наследником класса *cls*;
- `issubclass(cls, base)` проверяет, является ли класс *cls* наследником класса *base*.



*** Примечание:** так как в Python всё есть объект и отсутствуют примитивные типы данных, как правило, термины «тип» и «класс» являются синонимами.

Наследование и полиморфизм

Полиморфизм



- **Полиморфизм** – это способность одинаковым образом обрабатывать данные разных типов.
- Полиморфизм является фундаментальным свойством системы типов.

- статическая непалиморфная типизация
- статическая полиморфная типизация
- *динамическая типизация*

- специальный полиморфизм
- параметрический полиморфизм
- *полиморфизм подтипов*
(полиморфизм включений)

Наследование и полиморфизм

Утиная типизация



- **Неявная типизация, латентная типизация или утиная типизация** (англ. Duck typing) — вид динамической типизации, при которой границы использования объекта определяются его текущим набором методов и свойств, в противоположность наследованию от определённого класса. То есть считается, что объект реализует **интерфейс**, если он содержит все методы этого интерфейса, независимо от связей в иерархии наследования и принадлежности к какому-либо конкретному классу.
- Название термина пошло от английского «duck test» («утиный тест»), который в оригинале звучит как:
«If it looks like a duck, swims like a duck and quacks like a duck, then it probably is a duck».
(«Если это выглядит как утка, плавает как утка и крякает как утка, то, вероятно, это утка».).

Смотрите наши уроки в видео формате

ITVDN.com



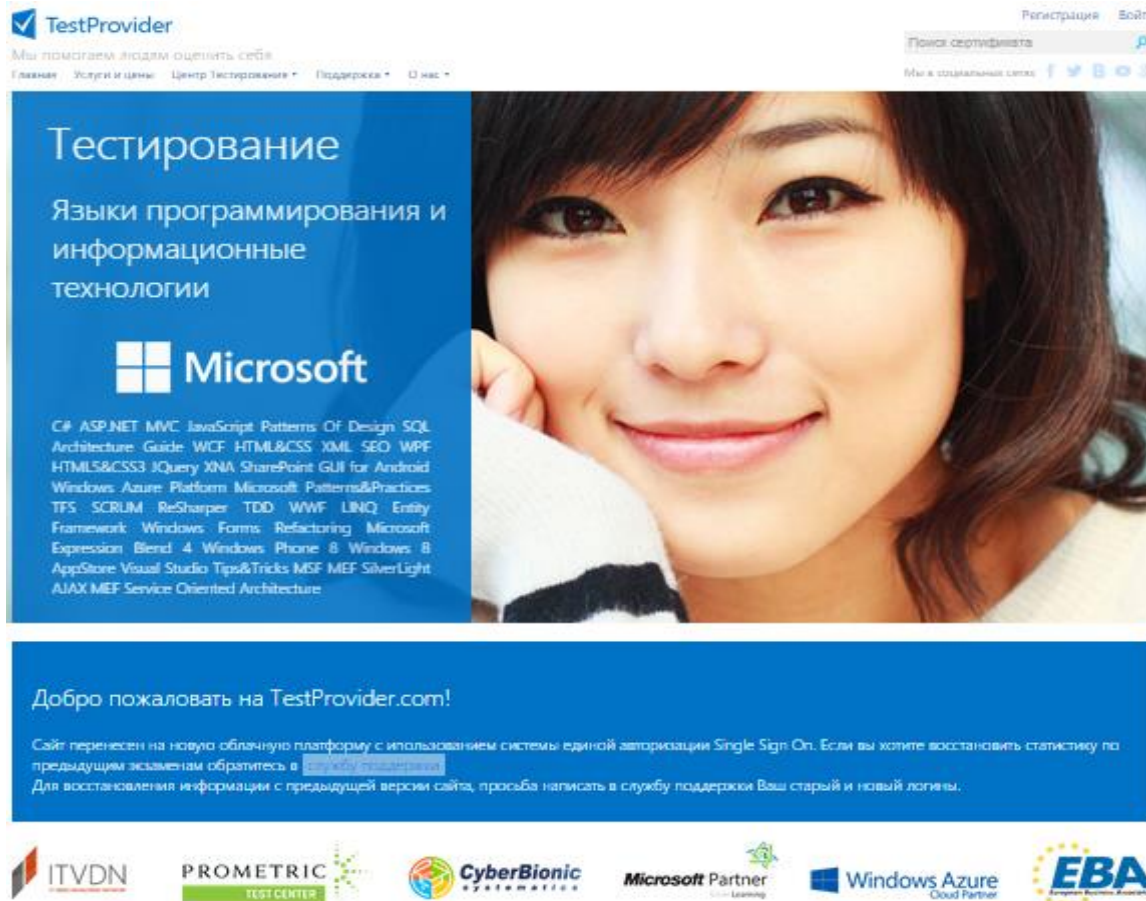
Посмотрите этот урок в видео формате на образовательном портале [ITVDN.com](http://itvdn.com) для закрепления пройденного материала.

Все курсы записаны сертифицированными тренерами, которые работают в учебном центре CyberBionic Systematics



Проверка знаний

TestProvider.com



TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и общей оценки знаний IT специалиста.

После каждого урока проходите тестирование для проверки знаний на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.



Python Essential

Q&A

Информационный видеосервис для разработчиков программного обеспечения

