Введение в объектно-ориентированное программирование

Парадигмы программирования

Парадигма программирования – это совокупность идей и понятий, определяющих стиль написания компьютерных программ, подход к программированию.

Python поддерживает разные парадигмы программирования

- императивное программирование
- процедурное программирование
- структурное программирование
- объектно-ориентированное программирование
- функциональное программирование

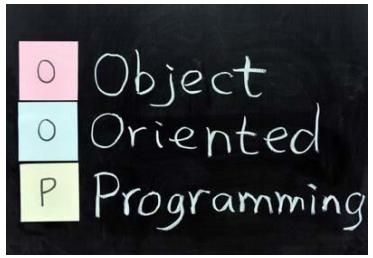
Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) – парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

Класс является моделью ещё не существующей сущности (объекта). Он является составным типом данным, включающим в себя поля и методы.
Объект – это экземпляр класса.

Основные принципы ООП:

- Абстракция
- Инкапсуляция
- Полиморфизм
- Наследование



Инкапсуляция

Инкапсуляция – это свойство системы, позволяющее объединиь данные и методы, работающие с ними, в классе, и скрыть детали реализации.

Инкапсуляция обеспечивается следующими средствами:

- контроль доступа
- методы доступа
- свойства объекта

ООП в Python

В Python всё является объектами – экземплярами каких-либо классов, даже сами классы, которые являются объектами – экземплярами метаклассов. Главным метаклассом является класс type, который является абстракцией понятия типа данных.

Всё есть объект

Классы в Python

- В терминологии Python члены класса называются атрибутами. Эти атрибуты могут быть как переменными, так и функциями.
- Классы создаются при помощи ключевого слова class.
- Классы как объекты поддерживают два вида операций: обращение к атрибутам классов и создание (инст анцирование) объектов – экземпляров класса (instance objects).
- Обращение к атрибутам какого-либо класса или объекта производится путём указания имени объекта и имени атрибута через точку.
- Для создания экземпляров класса используется синтаксис вызова функции.

Экземпляры классов в Python

- Единственная доступная операция для объектов-экземпляров – это доступ к их атрибутам.
- Атрибуты объект ов-экземпляров делятся на два типа: атрибуты-данные и методы.
- Атрибуты-данные аналогичны полям в терминологии большинства широко распространённых языков программирования.
- Атрибуты-данные не нужно описывать: как и переменные, они создаются в момент первого присваивания. Как правило, их создают в методе-конструкторе __init__ .
- Метод – это функция, принадлежащая объекту. Все атрибуты класса, являющиеся функциями, описывают соответствующие методы его экземпляров, однако они не являются одним и тем же.
- Особенностью методов является то, что в качестве первого аргумента им передаёт ся данный экземпляр класса. Таким образом, если obj – экземпляр класса MyClass, вызов метода obj.method() эквивалентен вызову функции MyClass.method(obj).

Первый аргумент метода, который соответствует текущему экземпляру, принято называть self.

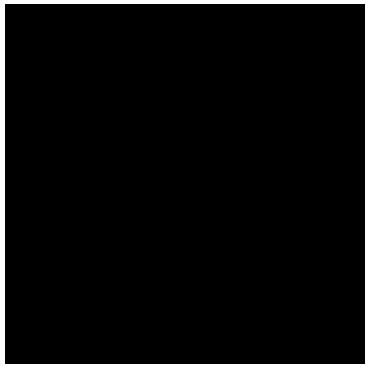
Разница между атрибутами класса и атрибутами-данными

Атрибуты класса являются общими для самого класса и всех его экземпляров. Их изменение отображается на все соответствующие объекты. Атрибуты-данные принадлежат конкретному экземпляру и их изменение никак не влияет на соответствующие атрибуты других экземпляров данного класса. Таким образом, атрибуты класса, которые не являются функциями, примерно соответствуют статическим полям в других языках программирования, а атрибуты-данные – обычным полям.

Следует понимать разницу между атрибутами класса и атрибутами-данными

Статические методы и методы класса

- Декоратор – это специальная функция, которая изменяет поведение функции или класса. Для применения декоратора следует перед соответствующим объявлением указать символ @, имя необходимого декоратора и список его аргументов в круглых скобках. Если передача параметров декоратору не требуется, скобки не указываются.
- Для создания статических методов используется декоратор staticmethod.
- Для создания методов класса используется декоратор classmethod.



Методы класса похожи на обычные методы, но относятся к самому классу как объекту – экземпляру метакласса (в отличие от обычных методов, которые принадлежат объектам – экземплярам классов, и статических методов, которые относятся к самому классу и всем его экземплярам и не принадлежат никакому объекту – экземпляру). Их первый аргумент принято называть cls.

Инкапсуляция в Python

- Все атрибуты по умолчанию являются публичными.
- Атрибуты, имена которых начинаются с одного знака подчёркивания (_) говорят программисту о том, что они относятся ко внутренней реализации класса и не должны использоваться извне, однако никак не защищены.
- Атрибуты, имена которых начинаются, но не заканчиваются, двумя символами подчёркивания, считаются приватными. К ним применяется механизм «name mangling». Он не предполагает защиты данных от изменения извне, так как к ним всё равно можно обратиться, зная имя класса и то, как Python изменяет имена приватных атрибутов, однако позволяет защитить их от случайного переопределения в классах-потомках.

Специальные атрибуты и методы

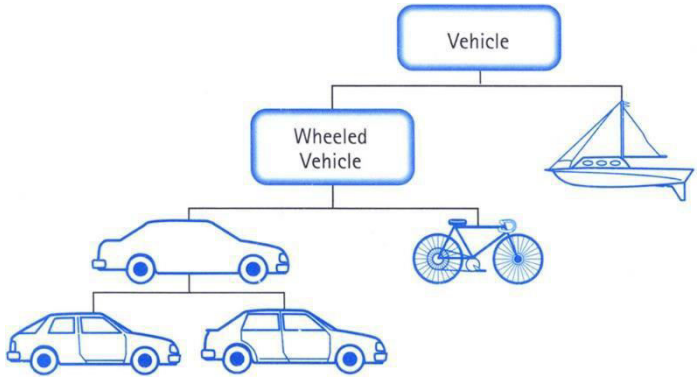
- Атрибуты, имена которых начинаются и заканчиваются двумя знаками подчёркивания, являются внутренними для Python и задают особые свойства объектов (примеры: __doc__ , __class__).
- Среди таких атрибутов есть методы. В документации Python подобные методы называются методами со специальными именами, однако в сообществе Python-разработчиков очень распространено название «магические методы». Также, встречается и название «специальные методы». Они задают особое поведение объектов и позволяют переопределять поведение встроенных функций и операторов для экземпляров данного класса.
- Наиболее часто используемым из специальных методов является метод __init__ , который автоматически вызывается после создания экземпляра класса.

Не следует объявлять свои собственные (нестандартные) атрибуты с именами, которые начинаются и заканчиваются двумя знаками подчёркивания

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.



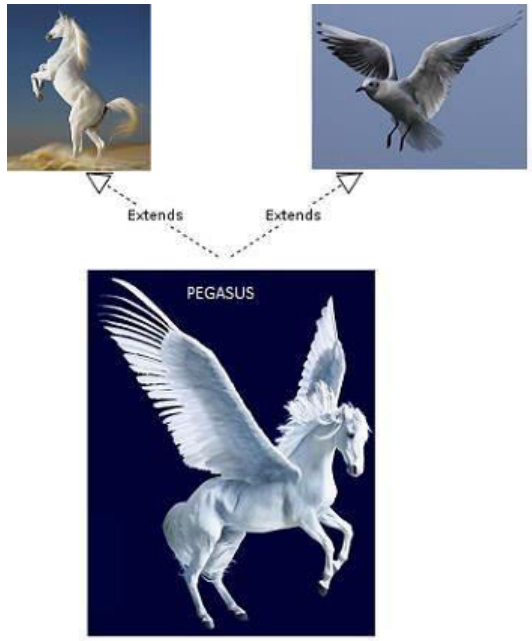
Классы старого и нового типа

- 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 т ипа (new-style classes) и классы ст арого т ипа (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.

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

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

Множественное наследование

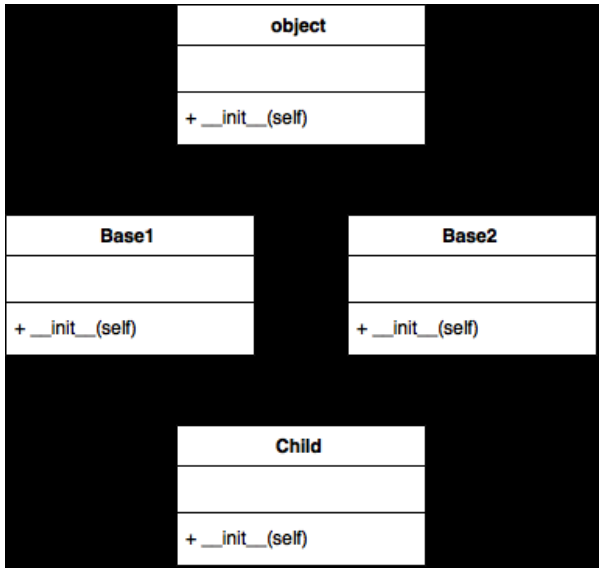


При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинство такого подхода в большей гибкости, однако он может быть потенциальным источником ошибок.

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

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

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()
```

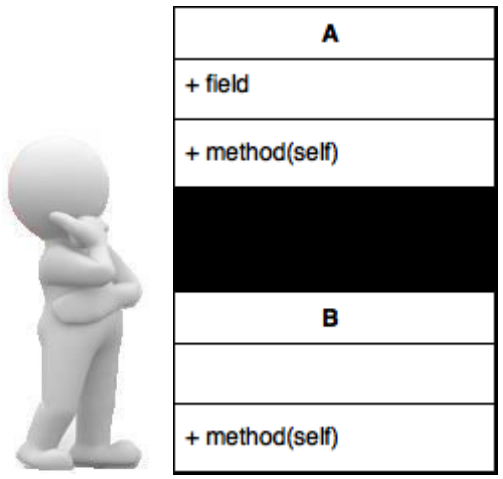
В Python 3 можно не указывать ничего и данные параметры будут получены автоматически:

```
super().method()    # то же самое, что super(__class__, self).method()
```

super недоступен для классов старого типа

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

Определение типа объекта



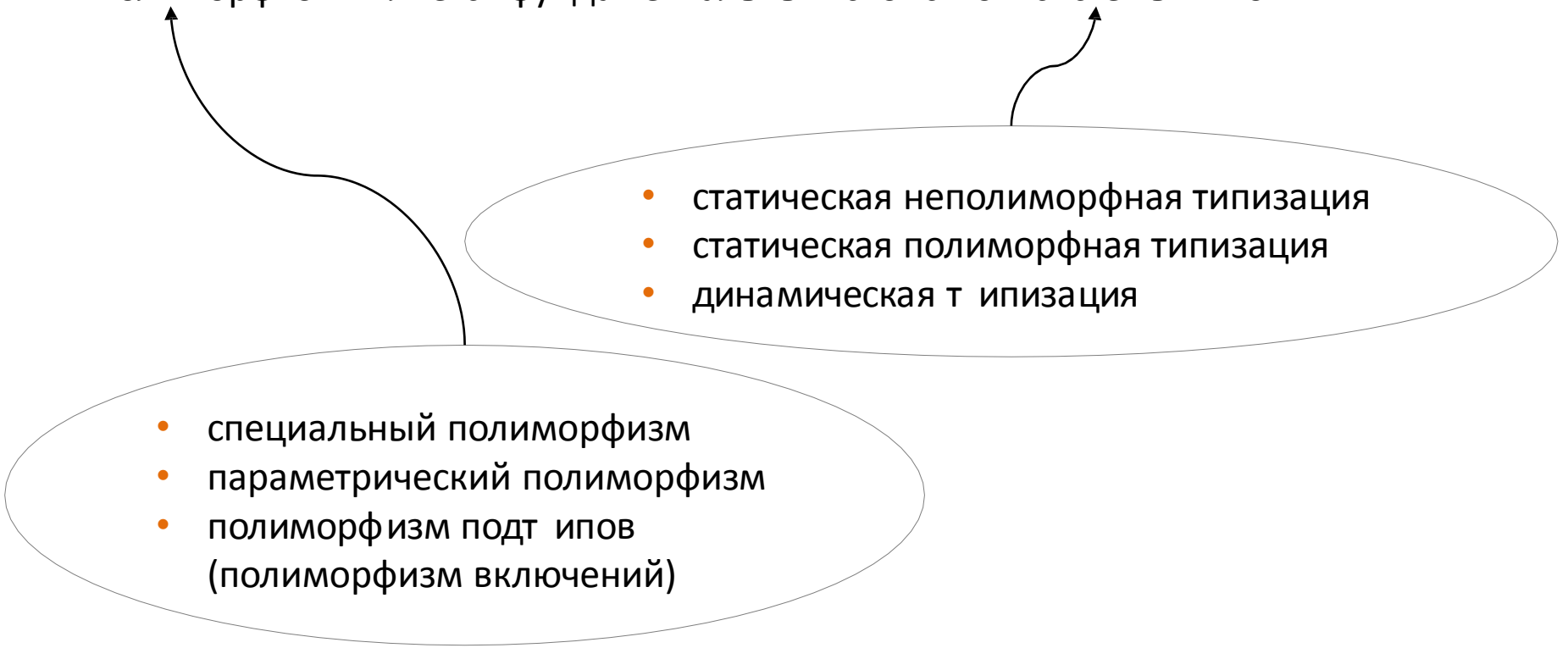
- Тип* данного объекта можно определить при помощи атрибута `class` и встроенной функции `type(obj)`.
- Атрибут класса `__bases__` хранит кортеж (неизменяемый список) базовых классов.
- Поскольку отношение наследования является транзитивным, в общем случае для проверки того, является ли данный объект 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.

Полиморфизм

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



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».).