Классы и объектно-ориентированное программирование

Стандартное поведение

- Оператор class создает объект класса и присваивает его имени. В точности как оператор def определения функции оператор class является исполняемым. После достижения и запуска он генерирует новый объект класса и присваивает его имени, указанному в заголовке class. Также подобно def операторы class обычно выполняются при первом импортировании файлов, где они находятся.lect-enable-clipboard t)

Стандартное поведение

- Присваивания внутри операторов class создают атрибуты классов. Как и в файлах модулей, присваивания на верхнем уровне внутри оператора class (не вложенные в def) генерируют атрибуты в объекте класса. Формально оператор class определяет локальную область видимости, которая превращается в пространство имен атрибутов для объекта класса подобно глобальной области видимости модуля. После выполнения оператора class атрибуты класса доступны посредством уточнения с помощью имени: объект.имя.

Стандартное поведение

- Присваивания внутри операторов class создают атрибуты классов. Как и в файлах модулей, присваивания на верхнем уровне внутри оператора class (не вложенные в def) генерируют атрибуты в объекте класса. Формально оператор class определяет локальную область видимости, которая превращается в пространство имен атрибутов для объекта класса подобно глобальной области ви димости модуля. После выполнения оператора class атрибуты класса доступны посредством уточнения с помощью имени: объект.имя.

Объекты экземпляров являются конкретными элементами

- Обращение к объекту класса как к функции создает новый объект экземпляра. При каждом обращении к классу он создает и возвращает новый объект экзем пляра. Экземпляры представляют конкретные элементы в предметной области программы.
- Каждый объект экземпляра наследует атрибуты класса и получает собственное пространство имен. Объекты экземпляров, созданные из классов, являются но выми пространствами имен. Объекты экземпляров начинают свое существование пустыми, но наследуют атрибуты, имеющиеся в объектах классов, из которых они были сгенерированы.
- Присваивания атрибутам аргумента self в методах создают атрибуты для от дельного экземпляра. Внутри функций методов класса первый аргумент (по соглашению называемый self) ссылается на обрабатываемый объект экземпляра; присваивания атрибутам аргумента self создают либо изменяют данные в экземпляре, но не в классе.

Объекты экземпляров

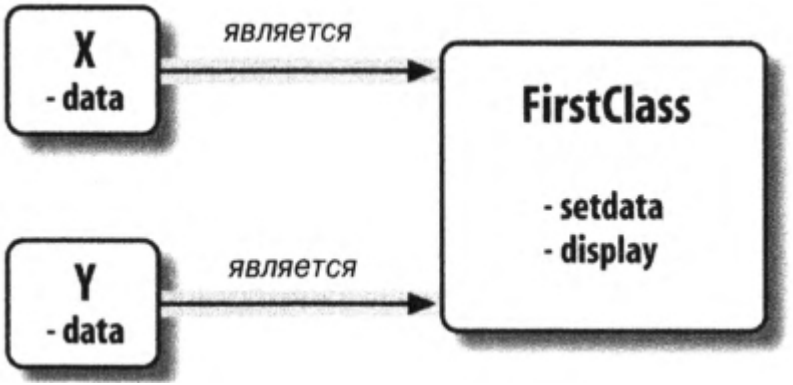
```
>>> class FirstClass:                  # Определить объект класса
        def setdata(self, value):      # Определить методы класса
            self.data = value          # self - это экземпляр
        def display(self):
            print(self.data)           # self.data: для каждого экземпляра
```

Функции внутри класса, как правило, называются методами. Они создаются пос редством нормальных операторов def и поддерживают все, что вам уже известно о функциях (т.е. могут иметь стандартные значения аргументов, возвращать значения, выдавать элементы по запросу и т.п.). Но первый аргумент в функции метода при ее вызове автоматически получает подразумеваемый объект экземпляра — объект, на ко тором произведен вызов.

Объекты экземпляров

```
>>> x = FirstClass()    # Создать два экземпляра
>>> y = FirstClass()    # Каждый представляет собой новое пространство имен
```

Обращаясь к классу таким способом (обратите внимание на круглые скобки), мы ге нерируем объекты экземпляров, представляющие собой просто пространства имен, которые имеют доступ к атрибутам своих классов. Собственно говоря, в этой точке мы имеем три объекта: два экземпляра и класс. В действительности мы располагаем тремя связанными пространствами имен. В терми нах ООП мы говорим, что экземпляр x “является” FirstClass, равно как и y — они оба наследуют имена, присоединенные к классу.



Объекты экземпляров

```
>>> x.setdata("King Arthur")   # Вызвать методы: self - это x
>>> y.setdata(3.14159)         # Выполняется FirstClass.setdata(y, 3.14159)
```

Ни x, ни y не имеет собственного атрибута setdata, поэтому чтобы найти его, Python следует по ссылке из экземпляра в класс. Вот и все, что нужно для наследования в Python: оно происходит во время уточнения атрибутов и предусматривает лишь поиск имен в связанных объектах — в данном случае за счет следования по ссылкам

В функции setdata класса FirstClass передаваемое значение присваивается self .data. Внутри метода self (имя, по соглашению назначаемое крайнему слева аргументу) автоматически ссылается на обрабатываемый экземпляр (x или y), так что присваивания сохраняют значения в пространствах имен экземпляров, а не класса;

```
>>> x.display()    # self.data отличается в каждом экземпляре
King Arthur
>>> y.display()    # Выполняется FirstClass.display(y)
3.14159
```

```
>>> x.data = "New value"     # Можно получать/устанавливать атрибуты
>>> x.display()              # И за пределами класса тоже
New value
```

```
>>> x.anothername = "spam"  # Здесь можно также устанавливать новые атрибуты!
```

Такой оператор присоединит к объекту экземпляра x новый атрибут по имени anothername, который может применяться или нет любым методом класса. Классы обычно создают все атрибуты экземпляра путем присваивания аргумента self, но они не обязаны поступать так — программы могут извлекать, изменять или создавать атрибуты для любых объектов, ссылками на которые они располагают.

наследование

- Помимо того, что классы служат фабриками для генерирования множества объектов экземпляров, они так же предоставляют возможность вносить изменения за счет ввода новых компонентов (называемых подклассами) вместо изменения существующих компонентов на месте.
- объекты экземпляров, сгенерированные из класса, наследуют атрибуты этого класса. Python также позволяет классам быть унаследованными от других классов, открывая возможность создания иерархий классов, которые специализируют поведение. Переопределяя атрибуты в подклассах, которые находятся ниже в иерархии, мы переопределяем более общие определения таких атрибутов выше в дереве. По сути, чем ниже мы углубляемся в иерархию, тем более специфическим становится программное обеспечение. Здесь отсутствуют какие-либо параллели с модулями, чьи атрибуты находятся в единственном плоском пространстве имен, которое неподдается настройке.

наследование

- Суперклассы перечисляются внутри круглых скобок в заголовке class. Чтобы заставить класс наследовать атрибуты от другого класса, просто укажите другой класс внутри круглых скобок в строке заголовка нового оператора class. Класс, выполняющий наследование, обычно называется подклассом, а класс, от которого производится наследование, является его суперклассом.
- Классы наследуют атрибуты от своих суперклассов. Точно так же, как экземпляры наследуют имена атрибутов, определенные в их классах, классы наследуют все имена атрибутов, которые определены в их суперклассах; при доступе к атрибутам Python находит их автоматически, если они не существуют в подклассах.
- Экземпляры наследуют атрибуты от всех доступных классов. Каждый экземпляр получает имена от класса, из которого он сгенерирован, а также от всех суперклассов этого класса. При поиске имени Python проверяет экземпляр, затем его класс и, наконец, все суперклассы.
- Каждая ссылка объект. атрибут инициирует новый независимый поиск. Python выполняет независимый поиск в дереве классов для каждого выражения с извлечением атрибута. Сюда входят ссылки на экземпляры и классы, сделанные за пределами операторов class (например, X.атрибут), а также ссылки на атрибуты экземпляра аргумента self в функциях методов класса. Каждое выражение self .атрибут в методе вызывает новый поиск для атрибута в self и выше.
- Изменения в логику вносятся за счет создания подклассов, а не модификации суперклассов. Переопределяя имена суперклассов в подклассах ниже в иерархии (дерева классов), подклассы замещают и тем самым настраивают унаследованное поведение.

Для чего используются классы?

- ***Инкапсуляция***

  это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя.

Для чего используются классы?

- ***Наследование***

  механизм объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом. Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

Для чего используются классы?

- **Полиморфизм**

  свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

  Например, если вы читаете данные из файла, то, очевидно, в классе, реализующем файловый поток, будет присутствовать метод похожий на следующий: byte[] readBytes(int n);

  Предположим теперь, что вам необходимо считывать те же данные из сокета. В классе, реализующем сокет, также будет присутствовать метод readBytes. Достаточно заменить в вашей системе объект одного класса на объект другого класса, и результат будет достигнут.

  При этом логика системы может быть реализована независимо от того, будут ли данные прочитаны из файла или получены по сети. Таким образом, мы абстрагируемся от конкретной специализации получения данных и работаем на уровне интерфейса. Единственное требование при этом – чтобы каждый используемый объект имел метод readBytes.

Множество экземпляров

- Классы по существу представляют собой фабрики для генерирования одного и более объектов. При каждом обращении к классу мы генерируем новый объект с отдельным пространством имен. Каждый объект, сгенерированный из класса,имеет доступ к атрибутам класса и получает собственное пространство имен для данных, которые варьируются от объекта к объекту. Методика похожа на сохранение состояния для каждого вызова функциями замыканий, но в классах она явная и естественная, к тому же отражает лишь одно из дел, обеспечиваемых классами. Классы предлагают завершенное программное решение.

Перегрузка операций

За счет предоставления специальных методов протокола классы могут определять объекты, реагирующие на всевозможные операции, которые мы видели в работе со встроенными типами. Скажем, к созданным с помощью классов объектам можно применять нарезание, конкатенацию, индексирование и т.д. Python предлагает привязки, которые классы могут использовать для перехвата и реализации любой операции для встроенных типов.