Цифровой практикум

ООП Python
Лекция 2. Основы написания классов

Баранов М.А.

Содержание лекции

- Генерация экземпляров;
- Объекты классов и их поведение;
- Объекты экземпляров;
- Пример;
- Наследование;
- Пример;
- Модули и атрибуты;
- Перехват операций;
- Пример;
- Остальное (optional)

Объекты классов обеспечивают 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.

• 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**.

• Атрибуты класса снабжают объект состоянием и поведением. Атрибуты объекта класса хранят информацию о состоянии и описывают поведение, которое разделяется всеми экземплярами, создаваемыми из класса; операторы **def** определения функций, вложенные внутрь **class**, генерируют методы, которые обрабатывают экземпляры.

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 экземпляра. Экземпляры представляют конкретные элементы в предметной области программы.

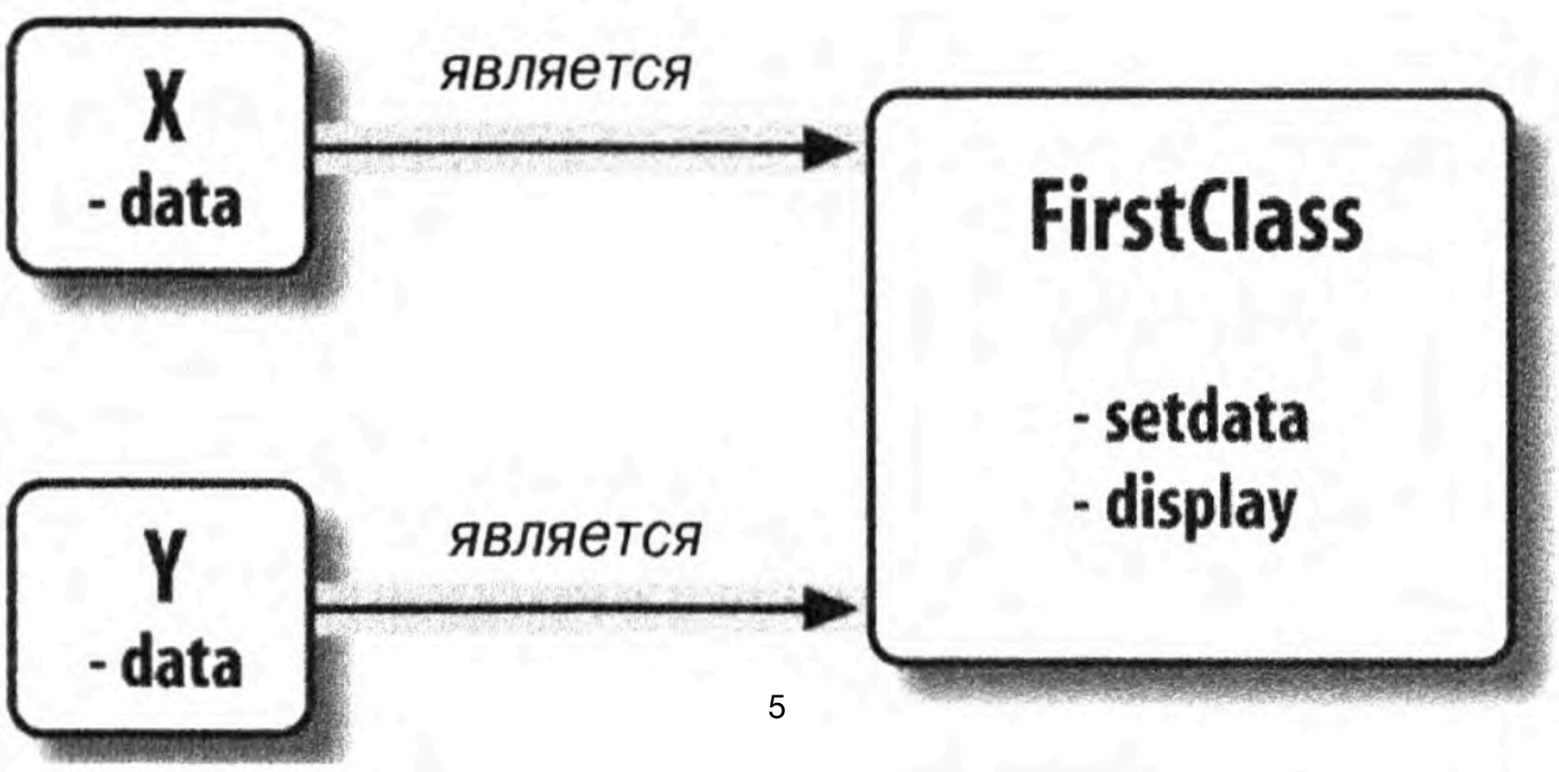
- Каждый объект экземпляра наследует атрибуты класса и получает собственное пространство имен. Объекты экземпляров, созданные из классов, являются новыми пространствами имен. Объекты экземпляров начинают свое существование пустыми, но наследуют атрибуты, имеющиеся в объектах классов, из которых они были 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.

Пример 1

```
>>> 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

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



Пример 1

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

>>> 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!
```

Классы настраиваются через наследование

- Чтобы заставить класс наследовать атрибуты от другого класса, просто укажите другой класс внутри круглых скобок в строке заголовка нового оператора **class**.

- Классы наследуют атрибуты от своих родительских классов (суперклассов);

- Экземпляры наследуют атрибуты от всех доступных классов;

- Каждая ссылка **объект.атрибут** инициирует новый независимый поиск. Python выполняет независимый поиск в дереве классов для каждого выражения с из влечением атрибута;

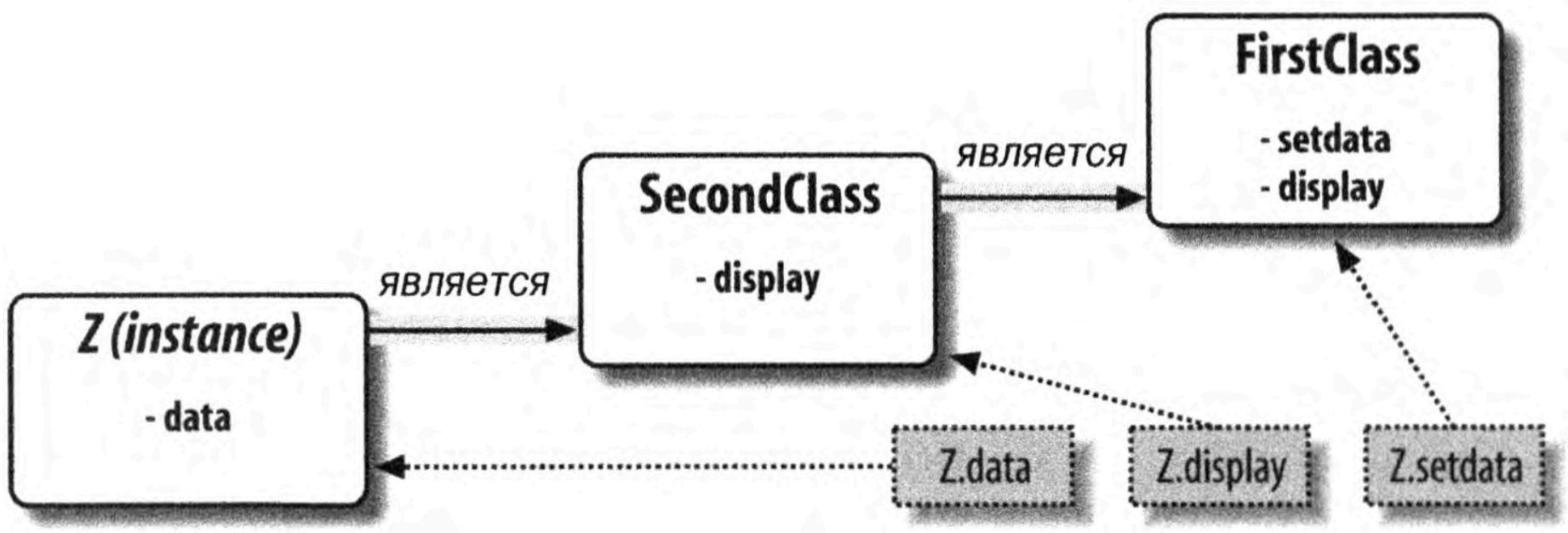
- Изменения логику вносятся за счёт создания подклассов, а не модификации суперклассов.

Пример 2

```
>>> class SecondClass(FirstClass):                    # Наследует setdata
        def display(self):                            # Изменяет display
            print('Current value = "%s"' % self.data)

>>> z = SecondClass()
>>> z.setdata(42)          # Находит setdata в FirstClass
>>> z.display()            # Находит переопределенный метод в SecondClass
Current value = "42"

>>> x.display() # x - по-прежнему экземпляр FirstClass (выводит старое сообщение)
New value
```



Классы являются атрибутами в модулях

```
from modulename import FirstClass    # Копировать имя в текущую область видимости
class SecondClass(FirstClass):       # Использовать имя класса напрямую
    def display(self): ...
```

Или вот эквивалент:

```
import modulename                                  # Доступ к целому модулю
class SecondClass(modulename.FirstClass):          # Уточнение для ссылки
    def display(self): ...
```

```
import person                        # Импортировать модуль
x = person.person()                  # Класс внутри модуля
```

```
from person import person            # Получить класс из модуля
x = person()                         # Использовать имя класса
```

Классы могут перехватывать операции Python

- Методы, имена которых содержат удвоенные символы подчеркивания (__ **X**__), являются специальными привязками. В классах Python мы реализуем перегрузку операций за счет предоставления особым образом именованных методов для перехвата операций.

- Такие методы вызываются автоматически, когда экземпляры встречаются во встроенных операциях. Скажем, если объект экземпляра наследует метод __ **add**__ , то этот метод вызывается всякий раз, когда объект появляется в выражении с операцией **+**.

- Классы могут переопределять большинство встроенных операций с типами. Существуют десятки специальных имен методов для перегрузки операций, которые можно перехватывать и реализовывать почти каждую операцию, действующую на встроенных типах.

- Для методов перегрузки операций не предусмотрены стандартные реализации и ни один из них не является обязательным.

- Классы нового стиля имеют ряд стандартных реализаций, но не для распространенных операций.

- Операции позволяют интегрировать классы в объектную модель Python.

Пример 3

- **__init__** выполняется, когда создается новый объект экземпляра: **self** является новым объектом **ThirdClass**;
- **__add__** выполняется, когда экземпляр **ThirdClass** присутствует в выражении **+**;
- **__str__** выполняется, когда объект выводится (формально при его преобразовании в отображаемую строку встроенной функцией **str** или ее внутренним эквивалентом Python).

```
>>> class ThirdClass(SecondClass):       # Унаследован от SecondClass
        def __init__(self, value):        # Вызывается для ThirdClass(value)
            self.data = value
        def __add__(self, other):         # Вызывается для self + other
            return ThirdClass(self.data + other)
        def __str__(self):                # Вызывается для print(self), str()
            return '[ThirdClass: %s]' % self.data
        def mul(self, other):             Изменение на месте: именованный метод
            self.data *= other
>>> a = ThirdClass('abc')                 # Вызывается __init__
>>> a.display()                           # Вызывается унаследованный метод
Current value = "abc"
>>> print(a)                              # __str__: возвращает отображаемую строку
[ThirdClass: abc]

>>> b = a + 'xyz'                         # __add__: создает новый экземпляр
>>> b.display()                           # b имеет все методы класса ThirdClass
Current value = "abcxyz"

>>> print(b)                              # __str__: возвращает отображаемую строку
[ThirdClass: abcxyz]

>>> a.mul(3)                              # mul: изменяет экземпляр на месте
>>> print(a)
[ThirdClass: abcabcabc]
```

Простейший класс Python

```
>>> class rec: pass          # Объект пустого пространства имен


>>> rec.name = 'Bob'         # Просто объект с атрибутами
>>> rec.age = 40


>>> print(rec.name)          # Подобен структуре C или записи
Bob

>>> x = rec()                # Экземпляры наследуют имена класса
>>> y = rec()


>>> x.name, y.name           # name хранится только в классе
('Bob', 'Bob')

>>> x.name = 'Sue'           # Но присваивание изменяет только x
>>> rec.name, x.name, y.name
('Bob', 'Sue', 'Bob')

>>> list(rec.__dict__.keys())
['__module__', '__dict__', '__weakref__', '__doc__', 'name', 'age']

>>> list(name for name in rec.__dict__ if not name.startswith('__'))
['age', 'name']
>>> list(x.__dict__.keys())
['name']
>>> list(y.__dict__.keys())  # list() не требуется в Python 2.X
[]
```

Простейший класс Python

```
>>> x.name, x.__dict__['name']  # Представленные здесь атрибуты являются
                                #   ключами словаря
('Sue', 'Sue')
>>> x.age                       # Но извлечение атрибута проверяет также классы
>>> x.__dict__['age']           # Индексирование словаря не производит поиск
                                #   в иерархии наследования
KeyError: 'age'
Ошибка ключа: 'age'

>>> x.__class__                 # Связь экземпляра с классом
<class '__main__.rec'>

>>> def uppername(obj):
        return obj.name.upper()    # По-прежнему необходим аргумент self (obj)

>>> uppername(x)                 # Вызов как простой функции
'SUE'

>>> rec.method = uppername       # Теперь это метод класса!

>>> x.method()                   # Запустить метод для обработки x
'SUE'

>>> y.method()                   # То же самое, но передать y для self
'BOB'

>>> rec.method(x)                # Можно вызывать через экземпляр или класс
'SUE'
```

Классы и словари

```
>>> rec = ('Bob', 40.5, ['dev', 'mgr']) # Запись на основе кортежа
>>> print(rec[0])
Bob

>>> rec = {}
>>> rec['name'] = 'Bob'                 # Запись на основе словаря
>>> rec['age'] = 40.5                   # Или {...}, dict(n=v) и т.д.
>>> rec['jobs'] = ['dev', 'mgr']
>>>
>>> print(rec['name'])
Bob

>>> class rec: pass
>>> rec.name = 'Bob'                    # Запись на основе класса
>>> rec.age = 40.5
>>> rec.jobs = ['dev', 'mgr']
>>>
>>> print(rec.name)
Bob
```

Классы и словари

```
>>> class rec: pass

>>> pers1 = rec()                          # Записи на основе экземпляров
>>> pers1.name = 'Bob'
>>> pers1.jobs = ['dev', 'mgr']
>>> pers1.age = 40.5
>>>
>>> pers2 = rec()
>>> pers2.name = 'Sue'
>>> pers2.jobs = ['dev', 'cto']
>>>
>>> pers1.name, pers2.name
('Bob', 'Sue')
```

Классы и словари

```
>>> class Person:
        def __init__(self, name, jobs, age=None):  # Класс = данные + логика
            self.name = name
            self.jobs = jobs
            self.age = age
        def info(self):
            return (self.name, self.jobs)

>>> rec1 = Person('Bob', ['dev', 'mgr'], 40.5)    # Вызовы конструктора
>>> rec2 = Person('Sue', ['dev', 'cto'])
>>>
>>> rec1.jobs, rec2.info()                        # Атрибуты + методы
(['dev', 'mgr'], ('Sue', ['dev', 'cto']))


>>> rec = dict(name='Bob', age=40.5, jobs=['dev', 'mgr'])       # Словари
>>> rec = {'name': 'Bob', 'age': 40.5, 'jobs': ['dev', 'mgr']}
>>> rec = Rec('Bob', 40.5, ['dev', 'mgr'])            # Именованные кортежи
```

Спасибо за внимание!