при поддержке
{а}вижн

Лекция 15

Перевод и адаптация материалов Джона ДеНиро (John DeNero). Используется с разрешения автора.

Объектно-ориентированное программирование

Объектно-ориентированное программирование

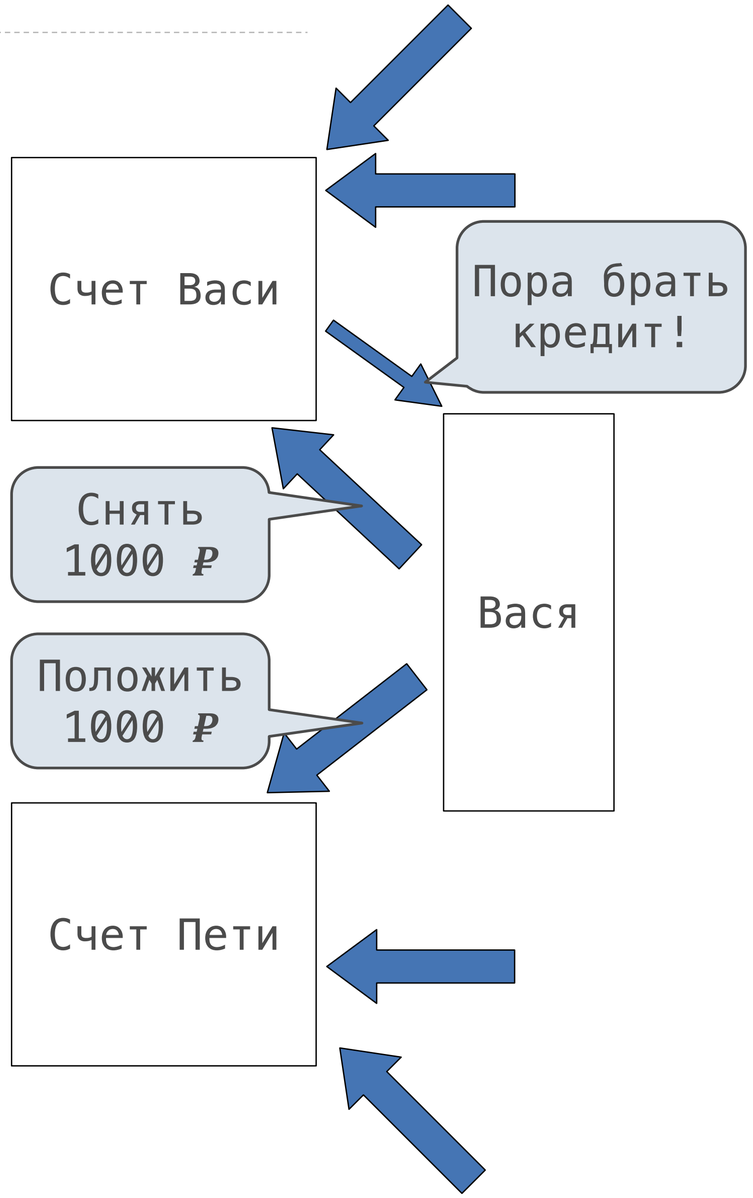
Способ модульной организации программ

- Границы абстракции
- Объединение информации и соответствующего поведения

Метафора для вычислений, использующая распределенное состояние

- Каждый объект имеет свое локальное состояние
- Каждый объект «знает» как управлять своим состоянием (методы)
- Вызовы методов — сообщения, посылаемые объектами друг другу
- Некоторые объекты могут быть экземплярами одного типа
- Различные типы ассоциированы друг с другом

- Специальный синтаксис и термины для поддержки метафоры



Классы

Класс является «шаблоном» для своих экземпляров

Идея: Все банковские счета включают баланс `balance` и владельца `holder`; класс `Account` должен добавлять соответствующие атрибуты каждому новому экземпляру.

Идея: Все банковские счета должны поддерживать «снятие» `withdraw` и «пополнение» `deposit`, которые работают одинаковым образом.

Ещё лучше: Все банковские счета имеют общие методы `withdraw` и `deposit`.

```
>>> a = Account('Вася')
>>> a.holder
'Вася'
>>> a.balance
0

>>> a.deposit(15)
15
>>> a.withdraw(10)
5
>>> a.balance
5
>>> a.withdraw(10)
‘Недостаточно средств'
```

Инструкция class

Инструкция Class

```
class <имя>:
    <набор>
```

Набор выполняется при выполнении инструкции class.

Инструкция class создает новый класс и связывает этот класс с <именем> в первом фрейме текущего окружения.

Инструкции присвоения и def внутри <набора> создают атрибуты класса (не имена во фреймах!)

```
>>> class Clown:
...     nose = 'большой и красный'
...     def dance():
...         return 'Спасибо, не надо!'
...
>>> Clown.nose
'большой и красный'
>>> Clown.dance()
'Спасибо, не надо!'
>>> Clown
<class '__main__.Clown'>
```

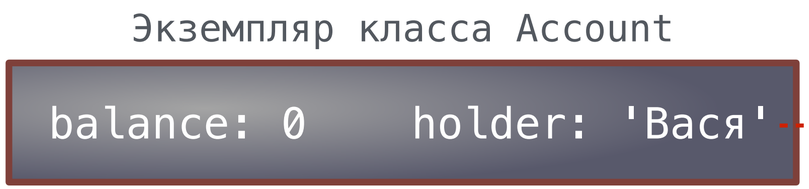
Создание объектов

Идея: Все банковские счета включают баланс `balance` и владельца `holder`; класс `Account` должен добавлять соответствующие атрибуты каждому новому экземпляру.

```
>>> a = Account('Вася')
>>> a.holder
'Вася'
>>> a.balance
0
```

При «вызове» класса:

1. Создается новый экземпляр класса:



2. Методу класса `__init__` при вызове, в качестве первого аргумента передается новый объект (`self`), вместе с ним передаются дополнительные аргументы, указанные в «вызове» класса.

```
class Account:
    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder
```

Метод `__init__` называют «конструктор»

Идентичность объектов

Каждый объект, являющийся экземпляром пользовательского класса, обладает уникальной идентичностью:

```
>>> a = Account('Вася')
>>> b = Account('Петя')
>>> a.balance
0
>>> b.holder
'Петя'
```

Каждый вызов Account создает новый экземпляр Account. Существует единственный класс Account.

Операторы идентичности «is» и «is not» проверяют, что значения двух выражений являются одним и тем же объектом:

```
>>> a is a
True
>>> a is not b
True
```

Связывание объекта с именем с помощью присвоений не порождает новые объекты:

```
>>> c = a
>>> c is a
True
```

Методы

Методы

Методы — это функции, определённые внутри класса

```
class Account:
    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance
    def withdraw(self, amount):
        if amount > self.balance:
            return ‘Недостаточно средств'
        self.balance = self.balance - amount
        return self.balance
```

self всегда связан с экземпляром класса Account

Эти инструкции def создают функции обычным образом, однако их имена добавляются не в текущий фрейм, а в список атрибутов класса.

Вызов методов

Все вызываемые методы «доступаются» до объекта через параметр `self`, так они получают доступ к данным объекта (состоянию).

```
class Account:
    ...
    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance
```

Определение с двумя параметрами

Нотация с точкой автоматически передает первый аргумент в метод.

```
>>> den_account = Account('Ден')
>>> den_account.deposit(100)
100
```

Связывается с self

Вызывается с единственным аргументом

Выражения с точкой

Объекты получают сообщения с помощью точечной нотации.

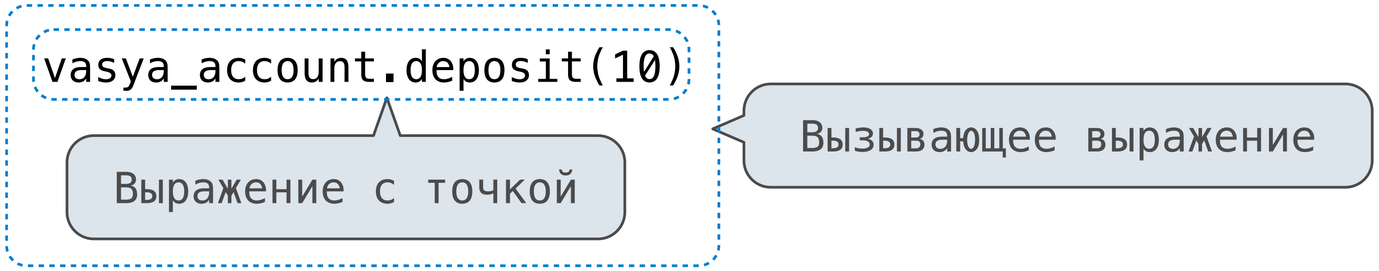
Точечная нотации позволяет обращаться к атрибутам экземпляра или его класса.

<выражение> . <имя>

<Выражение> может быть любым корректным выражением на Python.

<Имя> должно быть просто именем...

которое возвращает значение атрибута, найденное по <имени> в объекте,
который является результатом <выражения>.



(Пример)

Атрибуты

(Пример)

Доступ к атрибутам

Используя `getattr`, можно отыскивать атрибуты по строковому имени

```
>>> getattr(vasya_account, 'balance')
10

>>> hasattr(vasya_account, 'deposit')
True
```

`getattr` и выражения с точкой ищут значение имени одинаковым образом

Поиск по имени атрибута в объекте может вернуть:

- один из атрибутов экземпляра, или
- один из атрибутов класса экземпляра

Методы и функции

В Python различают:

- *Функции* — они рассматриваются с начала курса и
- *Связанные методы* — объединяют функции с объектами, в которых осуществляется вызов.

Объект + Функция = Связанный метод

```
>>> type(Account.deposit)
<class 'function'>
>>> type(vasya_account.deposit)
<class 'method'>
```

```
>>> Account.deposit(vasya_account, 656)
666
>>> vasya_account.deposit(671)
1337
```

Функция: все аргументы в скобочках

Метод: Один объект до точки и потом аргументы в скобочках

Поиск атрибутов по имени

<выражение> . <имя>

При выполнении выражения с точкой:

1. Выполнить <выражение> слева от точки, которое вернёт объект.
2. В атрибутах объекта ищется совпадение <имени>; если атрибут с таким именем существует, то возвращается его значение.
3. В противном случае, <имя> ищется в классе с возвратом значения атрибута.
4. Если это функция, то вместо функции Python вернет связанный метод.

Атрибуты класса

Атрибуты класса являются «общими» для всех экземпляров этого класса, поскольку это атрибуты класса, а не конкретного экземпляра.

```
class Account:

    interest = 0.02   # Атрибут класса

    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    # Здесь определены дополнительные методы


>>> vasya_account = Account('Вася')
>>> petya_account = Account('Петя')
>>> vasya_account.interest
0.02
>>> petya_account.interest
0.02
```

Атрибут **interest** не является ***частью*** экземпляра — это часть класса!

Присвоение атрибутам

Присвоение атрибутам

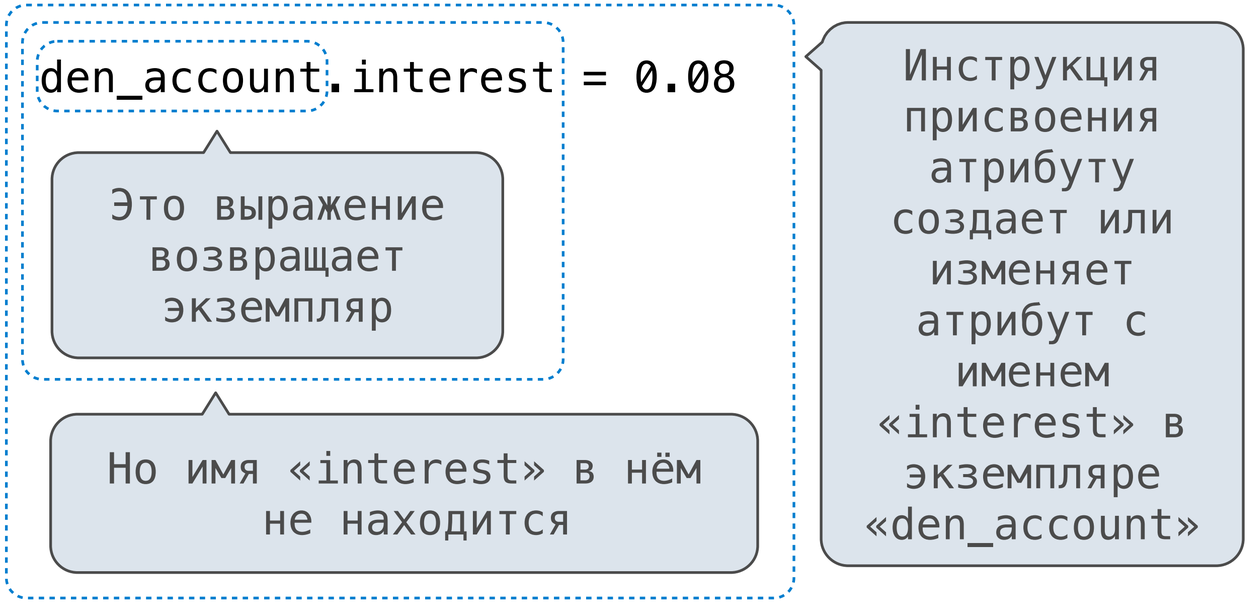
Инструкция присвоения выражению с точкой слева от знака равенства изменяет атрибут объекта указанного в выражении с точкой.

- Если объект — это экземпляр, то присвоение изменяет атрибут экземпляра
- Если объект — это класс, то присвоение изменяет атрибут класса

```
class Account:
    interest = 0.02
    def __init__(self, holder):
        self.holder = holder
        self.balance = 0
    ...

den_account = Account('Ден')
```

Присвоение атрибуту экземпляра :



Присвоение атрибуту класса :

```
Account.interest = 0.04
```

Инструкция присвоения атрибуту

Атрибуты класса Account

```
interest: 0.02  0.04  0.05
(withdraw, deposit, __init__)
```

Атрибуты экземпляра vasya_account

```
balance:  0
holder:   'Вася'
interest: 0.08
```

Атрибуты экземпляра petya_account

```
balance:  0
holder:   'Петя'
```

```
>>> vasya_account = Account('Вася')
>>> petya_account = Account('Петя')
>>> petya_account.interest
0.02
>>> vasya_account.interest
0.02
>>> Account.interest = 0.04
>>> petya_account.interest
0.04
>>> vasya_account.interest
0.04
```

```
>>> vasya_account.interest = 0.08
>>> vasya_account.interest
0.08
>>> petya_account.interest
0.04
>>> Account.interest = 0.05
>>> petya_account.interest
0.05
>>> vasya_account.interest
0.08
```