Object Oriented Programming Through Python (2.7.x)

Niko Laaksonen

November 19, 2014

What is OOP?

- The concept of an object is central in OOP (surprisingly)

What is OOP?

- The concept of an object is central in OOP (surprisingly)
  - attributes (data)

What is OOP?

- The concept of an object is central in OOP (surprisingly)
    - attributes (data)
    - methods (functions)

What is OOP?

- The concept of an object is central in OOP (surprisingly)
  - attributes (data)
  - methods (functions)
  - e.g. `numbers.Complex`

What is OOP?

- The concept of an object is central in OOP (surprisingly)
    - attributes (data)
    - methods (functions)
    - e.g. `numbers.Complex`
        - attributes: `Complex.real` `Complex.imag`

What is OOP?

- The concept of an object is central in OOP (surprisingly)
    - attributes (data)
    - methods (functions)
    - e.g. `numbers.Complex`
        - attributes: `Complex.real` `Complex.imag`
        - methods: `Complex.conjugate()`

What is OOP?

- The concept of an object is central in OOP (surprisingly)
  - attributes (data)
  - methods (functions)
  - e.g. `numbers.Complex`
    - attributes: `Complex.real` `Complex.imag`
    - methods: `Complex.conjugate()`

The OOP style

Objects interacting with each other through their methods.

OOP in Python

- In python this is implemented through classes

OOP in Python

- In python this is implemented through classes
    - A class is a description of the structure and functionality of the desired object (abstract)

OOP in Python

- In python this is implemented through classes
  - A class is a description of the structure and functionality of the desired object (abstract)
  - Objects are *instances* of classes (concrete)

OOP in Python

- In python this is implemented through classes
  - A class is a description of the structure and functionality of the desired object (abstract)
  - Objects are *instances* of classes (concrete)
- You can have multiple instances of the same class!

```
> z1 = complex(5,3)
> z2 = complex(1,2)
> print(z1 * z2.conjugate())
(11-7j)
```

Toy Example - Inheritance

```
class Animal():
   def __init__(self):
      self.legs = 0
      self.hasTail = False

   def speak(self):
      pass
```

Toy Example - Inheritance

```
class Animal():
   def __init__(self):
      self.legs = 0
      self.hasTail = False

   def speak(self):
      pass

class Cat(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 4
      self.hasTail = True

   def speak(self):
      print("Meow!")
```

Toy Example - Inheritance

```
class Animal():
   def __init__(self):
      self.legs = 0
      self.hasTail = False

   def speak(self):
      pass

class Cat(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 4
      self.hasTail = True

   def speak(self):
      print("Meow!")
```

```
class Bird(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 2
      self.hasTail = False

   def speak(self):
      print("Chirp!")
```

Toy Example - Inheritance

```
class Animal():
   def __init__(self):
      self.legs = 0
      self.hasTail = False

   def speak(self):
      pass

class Cat(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 4
      self.hasTail = True

   def speak(self):
      print("Meow!")
```

```
class Bird(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 2
      self.hasTail = False

   def speak(self):
      print("Chirp!")


> tim, marty = Cat(), Bird()
```

Toy Example - Inheritance

```
class Animal():
   def __init__(self):
      self.legs = 0
      self.hasTail = False

   def speak(self):
      pass

class Cat(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 4
      self.hasTail = True

   def speak(self):
      print("Meow!")
```

```
class Bird(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 2
      self.hasTail = False

   def speak(self):
      print("Chirp!")


> tim, marty = Cat(), Bird()
> tim.legs
4
```

Toy Example - Inheritance

```
class Animal():
   def __init__(self):
      self.legs = 0
      self.hasTail = False

   def speak(self):
      pass

class Cat(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 4
      self.hasTail = True

   def speak(self):
      print("Meow!")
```

```
class Bird(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 2
      self.hasTail = False

   def speak(self):
      print("Chirp!")


> tim, marty = Cat(), Bird()
> tim.legs
4
> marty.speak()
Chirp!
```

Toy Example - Inheritance

```
class Animal():
   def __init__(self):
      self.legs = 0
      self.hasTail = False

   def speak(self):
      pass

class Cat(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 4
      self.hasTail = True

   def speak(self):
      print("Meow!")
```

```
class Bird(Animal):
   def __init__(self):
      Animal.__init__(self)
      self.legs = 2
      self.hasTail = False

   def speak(self):
      print("Chirp!")


> tim, marty = Cat(), Bird()
> tim.legs
4
> marty.speak()
Chirp!
> tim.legs = 3
> #Tim the cat got injured :(
```

Advantages of Inheritance

- Avoids having to repeat the same code for multiple similar objects

Advantages of Inheritance

- Avoids having to repeat the same code for multiple similar objects
- Classes can inherit from more than one class, called multiple inheritance

Advantages of Inheritance

- Avoids having to repeat the same code for multiple similar objects
- Classes can inherit from more than one class, called multiple inheritance
    - Try to avoid!

Inheritance and Constructors

Suppose each Animal now keeps track of its age:

```
class Animal():
   def __init__(self, age=0):
      self.age = age
```

Inheritance and Constructors

Suppose each Animal now keeps track of its age:

```
class Animal():
   def __init__(self, age=0):
      self.age = age
```

```
> larry = Animal(3)
> bob = Animal(age=4)
```

Inheritance and Constructors

Suppose each Animal now keeps track of its age:

```
class Animal():
    def __init__(self, age=0):
        self.age = age
```

```
> larry = Animal(3)
> bob = Animal(age=4)
> larry = Cat(4)
TypeError: __init__() takes exactly 1 argument (2 given)
```

Inheritance and Constructors

Suppose each Animal now keeps track of its age:

```
class Animal():
    def __init__(self, age=0):
        self.age = age
```

```
> larry = Animal(3)
> bob = Animal(age=4)
> larry = Cat(4)
TypeError: __init__() takes exactly 1 argument (2 given)
```

```
class Cat(Animal):
    def __init__(self, *args, **kwargs):
        Animal.__init__(self, *args, **kwargs)
```

Key Ideas for OOP

- Not everything needs to be a class!

Key Ideas for OOP

- Not everything needs to be a class!
- Encapsulation is important

Key Ideas for OOP

- Not everything needs to be a class!
- Encapsulation is important
- The goal is to achieve good re-usability of the code

Key Ideas for OOP

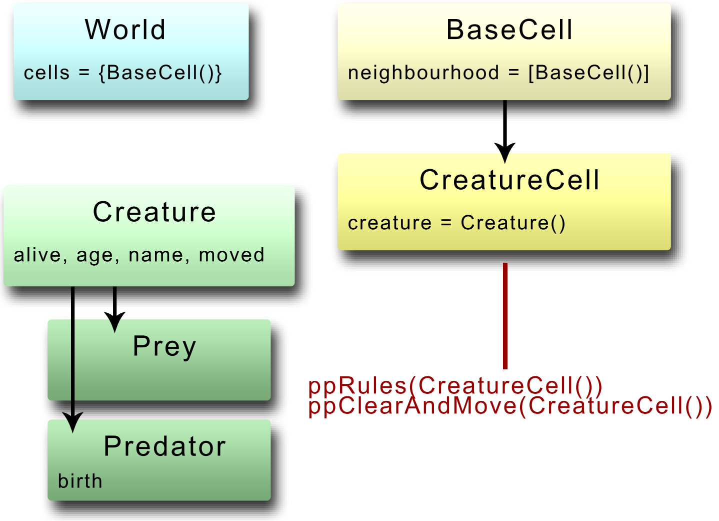
- Not everything needs to be a class!
- Encapsulation is important
- The goal is to achieve good re-usability of the code
- Polymorphism (we can write code for base class)

Real Example: Predator–prey system

`predator.py`

Real Example: Predator–prey system

predator.py



Comparison with C++

- `public`, `private`, `protected`

Comparison with C++

- `public`, `private`, `protected`
- Destructors

Comparison with C++

- `public`, `private`, `protected`
- Destructors
- Virtual functions

Comparison with C++

- `public`, `private`, `protected`
- Destructors
- Virtual functions
- Static vs. dynamic typing

Exercises

- Write classes and functions to represent our number system (integers, rational numbers).

Exercises

- Write classes and functions to represent our number system (integers, rational numbers).
- Can you think of a better ruleset for the predator–prey system?

Exercises

- Write classes and functions to represent our number system (integers, rational numbers).
- Can you think of a better ruleset for the predator–prey system?
- How would you keep track of the number of live predators and prey in each iteration?

Exercises

- Write classes and functions to represent our number system (integers, rational numbers).
- Can you think of a better ruleset for the predator–prey system?
- How would you keep track of the number of live predators and prey in each iteration?
- Export that data into a file and plot with your favourite plotting software. Do your results agree with the Lotka–Volterra equations?

Exercises

- Write classes and functions to represent our number system (integers, rational numbers).
- Can you think of a better ruleset for the predator–prey system?
- How would you keep track of the number of live predators and prey in each iteration?
- Export that data into a file and plot with your favourite plotting software. Do your results agree with the Lotka–Volterra equations?
- Write a new subclass for BaseCell to follow the rules of Conway's Game of Life.