

# Object Oriented Programming

Luis Pedro Coelho

§

On the web: <http://luispedro.org>

On twitter: @luispedrocoelho

European Molecular Biology Laboratory

June 16, 2014



# Procedural Programming

**Procedural programming:** organising programs around functions.

**Object-oriented programming:** organising programs around objects.

## OOP

**Aggregation** organise functions & data into classes.

**Encapsulation** hide information inside methods.

**Polymorphism** re-use code for multiple types.

**Inheritance** re-use code from one class to build another.

## Built-in Types

- 1 lists
- 2 dictionaries
- 3 strings
- 4 ...

## What's a Type

- 1 A domain of values
- 2 A set of methods (functions)

## List

- ① Domain: lists
- ② Functions: `L.append(e)`, `L.insert(idx,e)`, ...
- ③ Operators: `L[0]`, `'Rita' in L`

## List

- 1 Domain: lists
- 2 Functions: `L.append(e)`, `L.insert(idx,e)`, ...
- 3 Operators: `L[0]`, `'Rita' in L`

## Integer

- 1 Domain:  $\dots, -2, 1, 0, 1, 2, \dots$
- 2 Operators: `A + B`, ...

Object-oriented programming languages allow us to define new types.



# Motivating Example

## Simple Simulation

- 1 Boat goes around the ocean
- 2 You can move it around

## Boat Class

We define a Boat class, with two values, latitude & longitude, and five methods:

- 1 move\_north, move\_south, move\_east, move\_west
- 2 distance

# Using our Boat

```
b = Boat()  
b2 = Boat()  
b.move_north(1.)  
b2.move_south(2.)  
print b.distance(b2)
```

## Class

A class aggregates data and functions that belong together.

## Interface

### Functions:

- 1 Constructor: Takes the initial adaptation value and sigma.
- 2 `move_*`: Moves the boat.
- 3 `distance(b)`: Computes the distance between two boats.

### Data elements:

- 1 `latitude`: Current latitude.
- 2 `longitude`: Current longitude.



# Calling Methods

## Defining a method

```
class Boat(object):  
    def __init__(self, lat=0, long=0):  
        self.latitude = lat  
        self.longitude = long  
  
    def move_north(self, dlat):  
        self.latitude += dlat
```

## Calling a Method

```
obj = Boat()
```

```
obj.method(arg1, arg2)
```

## OOP

**Aggregation** organise functions & data into classes.

**Encapsulation** hide information inside methods.

**Polymorphism** re-use code for multiple types.

**Inheritance** re-use code from one class to build another.



## Type Polymorphism

Code is **polymorphic** if it can use different types without change

- A scientific boat can take samples as well as travel

```
class ScientificBoat(object):  
    def __init__(self, lat=0, long=0):  
        self.latitude = lat  
        self.longitude = long  
  
    def move_north(self, dlat):  
        ...
```

# Duck Typing



## OOP

**Aggregation** organise functions & data into classes.

**Encapsulation** hide information inside methods.

**Polymorphism** re-use code for multiple types.

**Inheritance** re-use code from one class to build another.

## Typical examples

- Actors in a simulation.
- File-like objects.
- Widgets.
- ...

The code for `ScientificBoat` is very similar to the code for `Boat`.

```
class ScientificBoat(Boat):
    '''
    A type of Boat, which can take samples
    '''
    '''
    def sample(self):
        ''' ... '''
```



# Lyskov Substitution Principle

If D inherits from C, then  
you should be able to use D anywhere you previously used C.

If D inherits from C, then  
D should behave-like C.

# New-Style vs. Old-Style Classes

```
class Boat(object):
```

```
    ...
```

Are we inheriting from `object`?

## OOP

**Aggregation** organise functions & data into classes.

**Encapsulation** hide information inside methods.

**Polymorphism** re-use code for multiple types.

**Inheritance** re-use code from one class to build another.