Programación orientada a objetos (OOP)

Lic. Rodrigo Lugones

rlugones@df.uba.ar

¿Qué es un struct?

Un Struct es una colección de variables. Son accesibles desde un único puntero. Internamente están contiguos en memoria.

Por ejemplo:

```
struct product {
  int weight;
  double price;
  string name;
} ;

product apple;
product banana, melon;

apple.price  = 10;
```

Google Python Style Guide

https://goo.gl/kxXVwK https://www.pylint.org/

Naming

link ▽ `module_name, package_name, ClassName, method_name, ExceptionName, function_name, GLOBAL_CONSTANT_NAME, global_var_name, instance_var_name, function_parameter_name, local_var_name.`

Names to Avoid

- single character names except for counters or iterators
- dashes (-) in any package/module name
- `__double_leading_and_trailing_underscore__` names (reserved by Python)

Naming Convention

- "Internal" means internal to a module or protected or private within a class.
- Prepending a single underscore (_) has some support for protecting module variables and functions (not included with `import * from`). Prepending a double underscore (__) to an instance variable or method effectively serves to make the variable or method private to its class (using name mangling).
- Place related classes and top-level functions together in a module. Unlike Java, there is no need to limit yourself to one class per module.
- Use CapWords for class names, but lower_with_under.py for module names. Although there are many existing modules named CapWords.py, this is now discouraged because it's confusing when the module happens to be named after a class. ("wait -- did I write `import StringIO` or `from StringIO import StringIO`?")

Guidelines derived from Guido's Recommendations

Type	Public	Internal
Packages	`lower_with_under`	
Modules	`lower_with_under`	`_lower_with_under`
Classes	`CapWords`	`_CapWords`
Exceptions	`CapWords`	
Functions	`lower_with_under()`	`_lower_with_under()`
Global/Class Constants	`CAPS_WITH_UNDER`	`_CAPS_WITH_UNDER`
Global/Class Variables	`lower_with_under`	`_lower_with_under`
Instance Variables	`lower_with_under`	`_lower_with_under (protected) or __lower_with_under (private)`
Method Names	`lower_with_under()`	`_lower_with_under() (protected) or __lower_with_under() (private)`
Function/Method Parameters	`lower_with_under`	
Local Variables	`lower_with_under`	

Main

▶ Even a file meant to be used as a script should be importable and a mere import should not have the side effect of executing the script's main functionality. The main functionality should be in a main() function.

WTPC Workshop en técnicas de programación científica

Programación orientada a objetos - OOP

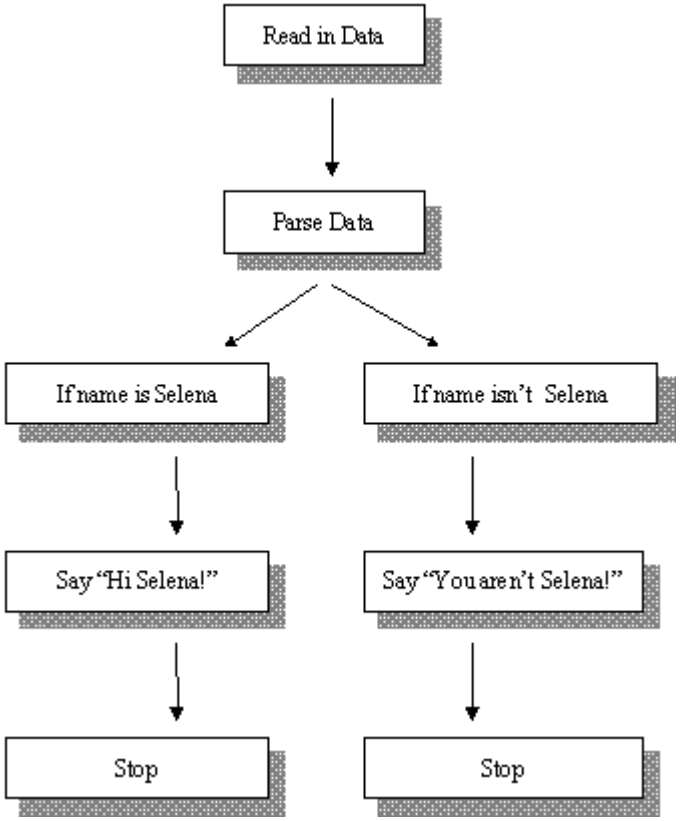
en Python

Los **objetos** son abstracciones de Python para referirse a los datos. Todos los datos en un programa de Python son representados por objetos o por relaciones entre objetos.

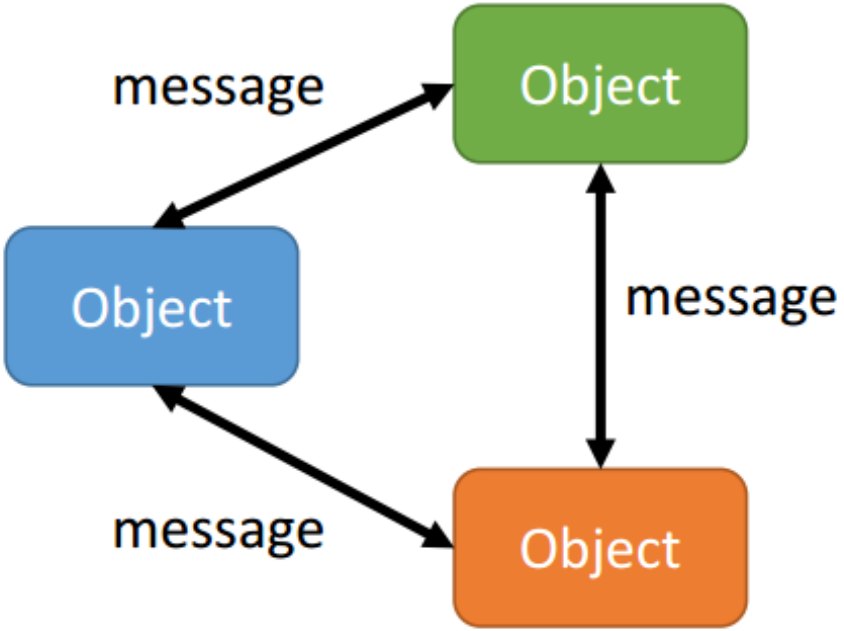
*https://docs.python.org/2/reference/datamodel.html#

Workflow procedural

1)in
2)process
3)out



Workflow en objetos



Clase

Un constructor de objetos

Estado

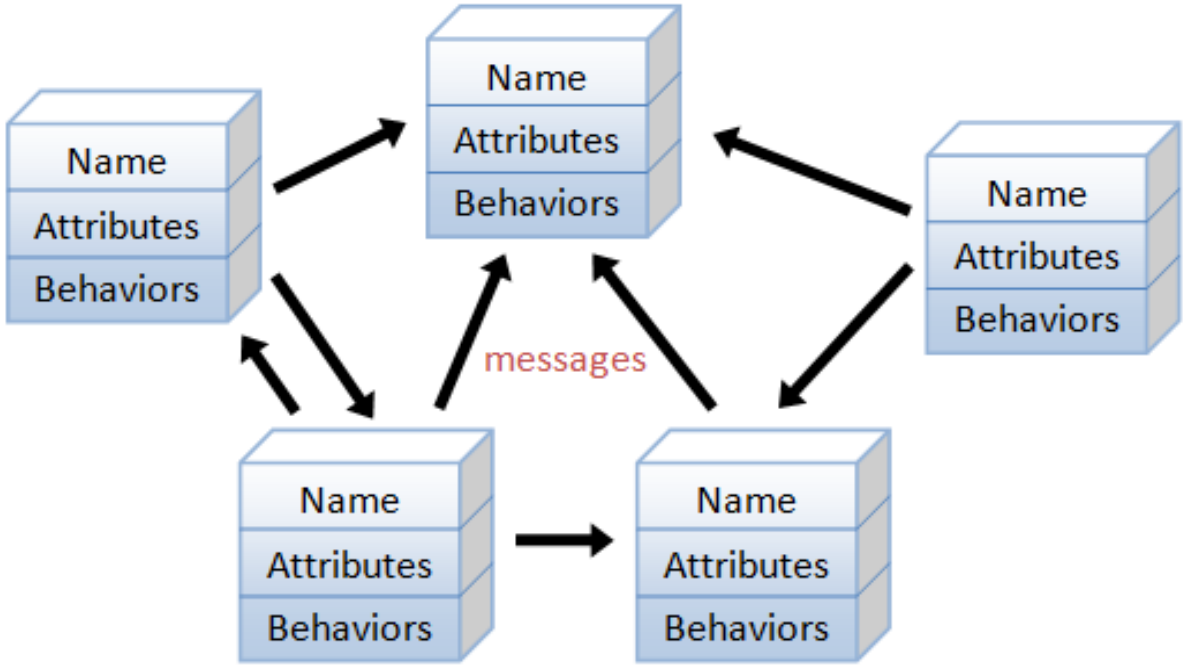
Todas las propiedades de un objeto

Comportamiento

Cómo un objeto reacciona frente a una interacción. Esto se logra llamando a ciertos métodos. En OOP es la manera en la que responde a ciertos mensajes.

Identidad

Distintos objetos pueden tener idénticos estados y el mismo comportamiento, pero cada uno tendrá su identidad.


Name
Attributes
Behaviors
Name
Attributes
Behaviors
Name
Attributes
Behaviors
messages
Name
Attributes
Behaviors
Name
Attributes
Behaviors

Programación orientada a Objetos

Composición

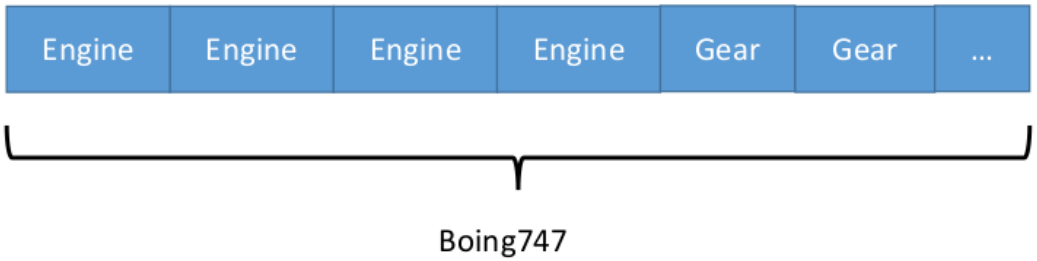
Encapsulación

Herencia

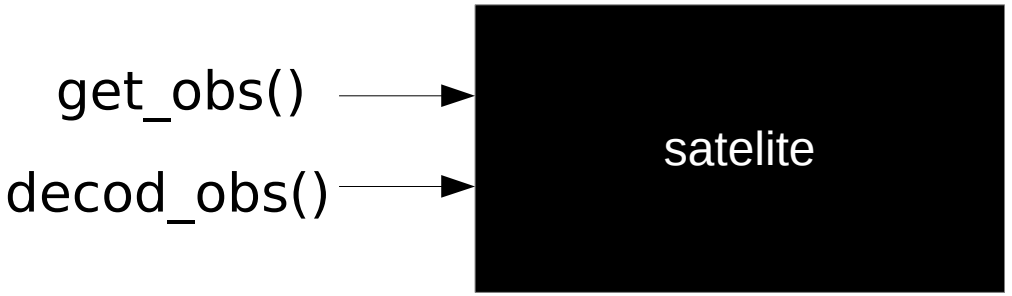
Polimorfismo

Composición

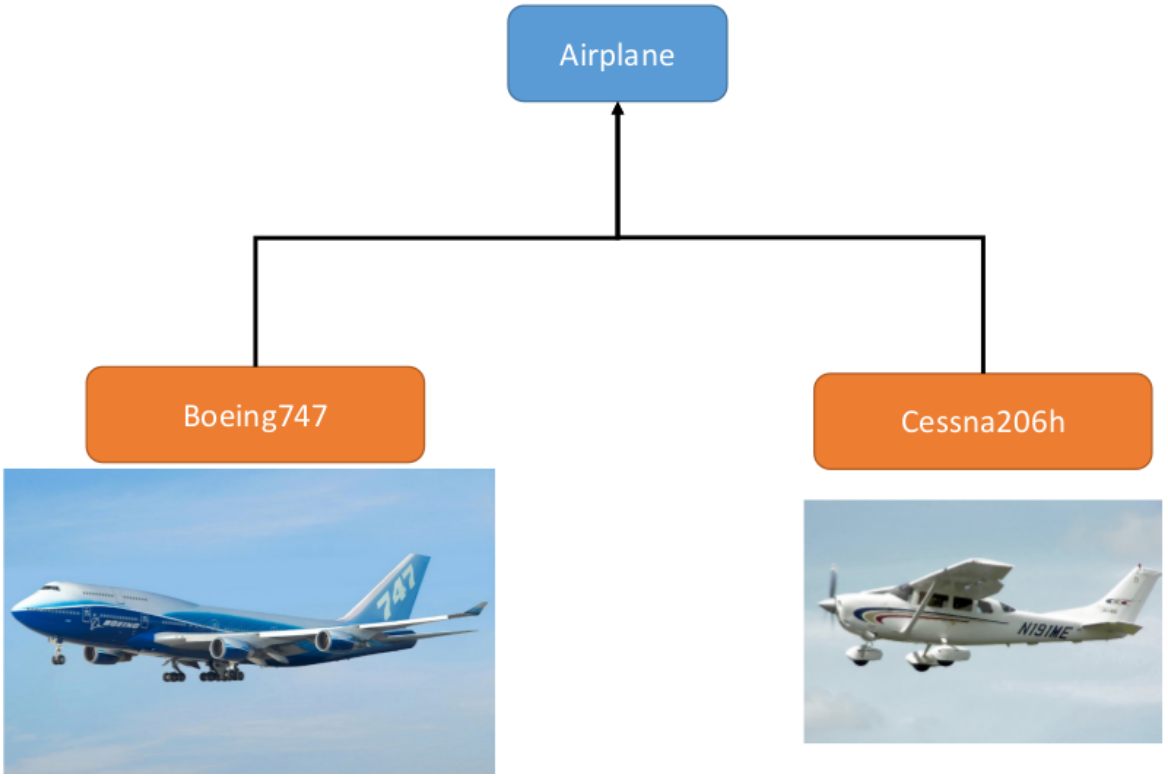
- natural way of creating new objects is by building them out of existing objects.
- Complex systems are composed out of simpler sub-systems



Encapsulación



Herencia



Polimorfismo

```
def decod_obs (date, satelites):
  for satelite in satelites:
      satelite.decod_obs(date)

def get_obs(date, satelites):
  for satelite in satelites:
      satelite.get_obs(date)
```

```
satelites = []
satelites.append(satellite1)
satelites.append(satellite2)
satelites.append(satellite3)

decod_obs (datetime(2016,3,8), satelites)
```

Clases

```
class TVseries(object):
    """
    Define a tv serie class
    """
    def __init__(self, name, eps):                    # constructor
        self.name = name                                    #  atributos, variables de clase
        self.eps_per_s = eps


    def status(self):                                      # método
        text = '{} has {} episodes per season.'
        return text.format(self.name, self.eps_per_s)
```

Clases

```
class TVseries(object):
    """
    Define a tv serie class
    """
    def __init__(self, name, eps):                    # constructor
        self.name = name                                    #  atributos, variables de clase
        self.eps_per_s = eps


    def status(self):                                      # método
        text = '{} has {} episodes per season.'
        return text.format(self.name, self.eps_per_s)
```

```
In [26]: got = TVseries('Game of Thrones', 10)
In [27]: bbt = TVseries('Big Bang Theory', 24)

In [28]: print bbt.name
Big Bang Theory

In [31]: print got.name
Game of Thrones

In [32]: print bbt.status()
Big Bang Theory has 24 episodes per season.

In [33]: print got.status()
Game of Thrones has 10 episodes per season.
```

Métodos

```
class TVseries(object):
    """
    Define a tv serie class
    """
    def __init__(self, name, eps):
        self.name = name
        self.eps_per_s = eps
        self.num_watched = 0

    def seen(self, num = 1):
      self.num_watched += num

    def status(self):
        text = '{} has {} episodes per season. I saw {} of them.'
        return text.format(self.name, self.eps_per_s, self.num_watched)
```

```
In [26]: got = TVseries('Game of Thrones', 10)
In [27]: bbt = TVseries('Big Bang Theory', 24)

In [28]: print bbt.name
Big Bang Theory

In [31]: bbt.seen(4)
In [32]: print bbt.status()
Big Bang Theory has 24 episodes per season.  I saw 4 of them.

In [33]: print got.status()
Big Bang Theory has 24 episodes per season.  I saw 0 of them.
```

Built-in methods

```
class TVseries(object):
  """
  Define a tv serie class
  """
  def __init__(self, name, eps):
    self.name = name
    self.eps_per_s = eps

 def seen(self,  num=1):
    self.num_watched  += num

  def __str__(self):
    text = '{} has {} episodes per season. I saw {} of them.'
     return text.format(self.name, self.eps_per_s , self.num_watched )
```

```
In [26]: got = TVseries('Game of Thrones', 10)
In [27]: bbt = TVseries('Big Bang Theory', 24)

In [28]: got.seen(4)
In [31]: print got

Game of Thrones has 10 episodes per season. I saw 0 of them.
```

Herencia

```
class Foo(object):
    def hello(self):
        print 'Hello! Foo here.'

    def bye(self):
        print 'Bye bye! (implemented in Foo)'


class Bar(Foo):
    def hello(self):
        print 'Hello! Bar here.'
```

```
In [2]: f =  Foo()
In [3]: b =  Bar()

In [4]: f.hello()
Hello! Foo here.

In [5]: f.bye()
Bye bye!  (implemented in Foo)

In [6]: b.hello()
Hello! Bar here.

In [7]: b.bye()
Bye bye!  (implemented in Foo)
```

Encapsulación

```
class Test(object):
    def __mangled_name(self):
        pass
    def normal_name(self):
        pass
```

```
In [3]: mi_test = Test()

In [4]: mi_test.__mangled_name()
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-4-b2d869da9175> in <module>()
----> 1 t.__mangled_name()

AttributeError: 'Test' object has no attribute '__mangled_name'
```

Copiando comportamiento

```
class Test(object):
        def __init__(self):
                self.val = 5 # immutable
                self.list = [5,6,7] # mutable
```

```
In [17]: a = Test()

In [18]: b = a
In [19]: c = copy(a)
In [20]: d = deepcopy(a)
In [21]: a.val, b.val, c.val, d.val
Out[21]: (5, 5, 5, 5)

In [22]: a.val = 7
In [23]: a.val, b.val, c.val, d.val
Out[23]: (7, 7, 5, 5)

In [24]: a.list, b.list, c.list, d.list
Out[24]: ([5, 6, 7], [5, 6, 7], [5, 6, 7], [5, 6, 7])

In [25]: a.list.append(999)
In [26]: a.list, b.list, c.list, d.list
Out[26]: ([5, 6, 7, 999], [5, 6, 7, 999], [5, 6, 7, 999], [5, 6, 7])

In [27]: a.list = 'Hello'
In [28]: a.list, b.list
Out[28]: ('Hello', 'Hello', [5, 6, 7, 999], [5, 6, 7])
```

```
sound/
        __init__.py
        formats/
                __init__.py
                wavread.py
                wavwrite.py
                aiffread.py
                aiffwrite.py
                auread.py
                auwrite.py
                ...
effects/
        __init__.py
        echo.py
        surround.py
        reverse.py
        ...
filters/
        __init__.py
        equalizer.py
        vocoder.py
        karaoke.py
```

```
import sound.effects as se

from sound.effects import echo

from sound.effects.echo import echofilter
```

Créditos y referencias

Modules & Objects
David Grellscheid
Workshop on Advanced Techniques for Scientific Programming and Management of Open Source Software Packages
ICPT SAIFR

Concepts of Object-Oriented Programming
Richard Berger richard.berger@jku.at
Johannes Kepler University Linz

WTPC
workshop en técnicas de programación científica

¿Preguntas?