CNRS ANF PYTHON

Objects everywhere

Marc Poinot

Numerical Simulation Dept.

marc.poinot@onera.fr

ONERA

THE FRENCH AEROSPACE LAB

retour sur innovation

Outline

- Python Object oriented features
  - Basic OO
    - concepts
    - syntax

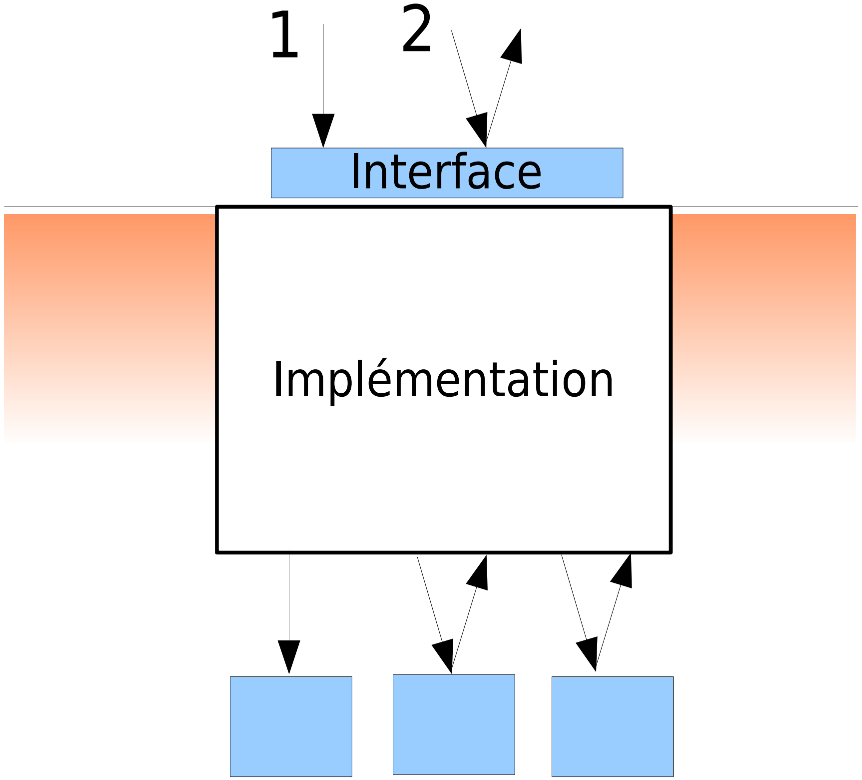
  - More on Python classes
    - multiple inheritance
    - reuse
    - introspection
    - mechanisms
      - decorators
      - operators
      - others

marc.poinot@onera.fr

Interface

An object is an interface

- Visible part of the object
  - Functions
  - Constants
  - Behavior
- Example:
  - Phone
- Encapsulation
- Isolation



marc.poinot@onera.fr

Python OO remarks

- No interface
  - Difficult to set a interface in a interpreted language
  - Run-time evaluation may change behaviour
  - Even if's very dangerous
  - No abstract classes

- No function signature
  - Cannot identify a method with its name & args

- Life cycle function specifics
  - Due to python object life cycle
  - Copy constructor, Destructeur, Init vs New

marc.poinot@onera.fr

Basics - 1

- A class is a type
  - An object is a value
  - A variable is a reference to a value
    - A class is a value

- syntax
  - **`class`** keyword
  - class scope variables are attribute
  - class scope functions are methods
    - mandatory first argument of method is the current object
    - by convention self variable name used for this first argument

Basics - 2

- Base class
  - The common part of all its subclasses
  - A subclass can be a base class
  - A derivation is the creation of a new class from a base class

- Methods overloading
  - Methods of same name are masking base's methods
  - Methods identification only uses the name, not the args

- Special methods
  - Methods with special names
  - Some methods are automatically called, for example:
    - create
    - delete
  - Operators
    - Never change the semantics of the overloaded operators

marc.poinot@onera.fr

Factorization & Derivation - 1

- Factorization
  - Same interface
    - Restrictions
    - Extensions

```
Object
  None
  Numbers
    Integers
        Plain integers
        Long integers
        Booleans
    Floating point
    Complex numbers
  Sequences
    Immutable sequences
        Strings
        Unicode
        Tuples
    Mutable sequences
        Lists
  Set types
    Sets
    Frozen sets
  Mappings
    Dictionaries
```

- A factorization is an arbitrary process
  - Target is the application
    - Sets are not Sequences
    - Integers are not Reals
    - Immutable sequences are not readonly Mutable sequences
  - Same values may lead
    - to different class hierarchy

marc.poinot@onera.fr

Factorization & Derivation - 2

- Two classes same interface

```
class ListeEntiers:

  def __init__(self):
    self.values=[]

  def add(self,v):
    self.values.append(v)

  def max(self):
    m=self.values[0]
    for v in self.values:
      if (v>m): m=v
    return m

  def count(self):
    return len(self.values)
```

```
class EnsembleEntiers:

  def __init__(self):
    self.values=[]

  def add(self,v):
    if (v not in self.values):
      self.values.append(v)

  def max(self):
    m=self.values[0]
    for v in self.values:
      if (v>m): m=v
    return m

  def count(self):
    return len(self.values)
```

```
l=ListeEntiers()
l.add(2)
l.add(3)
l.add(2)
print l.max(), l.count()


e=EnsembleEntiers()
e.add(2)
e.add(3)
e.add(2)
print e.max(), e.count()
```

marc.poinot@onera.fr

Factorization & Derivation - 3

- Factorization

```
class ListeEntiers:

  def __init__(self):
    self.values=[]

  def add(self,v):
    self.values.append(v)

  def max(self):
    m=self.values[0]
    for v in self.values:
      if (v>m): m=v
    return m

  def count(self):
    return len(self.values)
```

```
class EnsembleEntiers(ListeEntiers):

  def add(self,v):
    if (v not in self.values):
      self.values.append(v)
```

marc.poinot@onera.fr

Factorization & Derivation - 4

- Specialization

```
class ListeEntiers:

  def __init__(self):
    self.values=[]

  def add(self,v):
    self.values.append(v)

  def max(self):
    m=self.values[0]
    for v in self.values:
      if (v>m): m=v
    return m

  def count(self):
    return len(self.values)

  def has(self,v):
    return v in self.values
```

```
class EnsembleEntiers(ListeEntiers):

  def add(self,v):
    if (v not in self.values):
      self.values.append(v)

  def remove(self,v):
    self.values.remove(v)
```

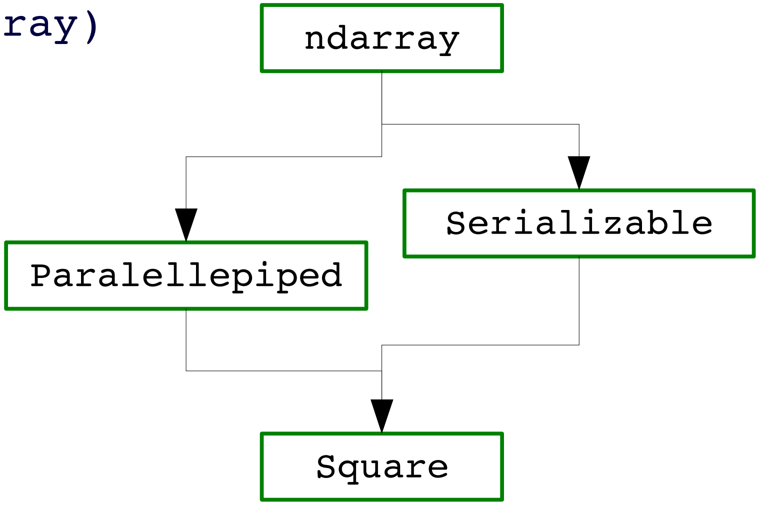
marc.poinot@onera.fr

Factorization & Derivation - 5

- Simple derivation

```
class Parallelepiped(numpy.ndarray)
class Serializable(object)
```

  - Use super() to find out base class



ndarray

Serializable

Paralellepiped

Square

- Multiple derivation

```
class Square(Parallelepiped,Serializable)
```

  - Method Resolution Order

```
Square, Parallelepiped, Serializable, ndarray, ..., object
```

  - Methods are called wrt the MRO (depth first)

Class indentification & introspection

- isinstance(O,C)
  - check object O is of class C or one of its base class
- issubclass(C,B)
  - check class C has B as base class

- Method resolution order

```
C.mro()
```

- module *inspect*
  - large set of function to parse and retrieve info

```
inspect.getmembers(parser, predicate=inspect.ismethod)
```

- Variable browsing
  - difficult to find back variables

```
dir(), globals(), locals()
```

marc.poinot@onera.fr

Methods

- method
  - callable class attribute
  - object as first arg
  - to be run on object values

`mesh.actualMemory()`

- classmethod
  - to be run on class values, no object required
  - class as first arg

`Mesh.requiredMemory()`

`@classmethod`

- staticmethod
  - no class no self as arg

`Mesh.getMaxMemory()`

`@staticmethod`

A property is a synctactic trick:

```
@property
def f():
  pass
```

marc.poinot@onera.fr

Special methods - 1

- init vs new
  - new(cls, *args)
    - creation
    - takes the class as arg
    - returns the new instance
    - ndarray: no init in order to be able to change actual class
  - init(self, *args)
    - initialisation
    - takes the new instance as arg
    - returns self
  - del
    - Refcount reaches zero

marc.poinot@onera.fr

Special methods - 2

- Isolate interface/ implementation
  - attribute maybe actual value, function, proxy
  - getter/setter usual pattern

```
class A(object):
   def __init__(self,*args):
      self.data=None

a=A()
a.data=4
v=a.data
```

```
class A(object):
   def __init__(self,*args):
      self._data=None

   def set_data(self,value)
      self._data=value

   def get_data(self):
      return self._data

a=A()
a.set_data(4)
```

```
class A(object):
   def __init__(self,*args):
      self._data=None

   @property
   def data(self):
     return self._data

   @data.setter
   def data(self,value)
      self.data=value

a=A()
a.data=4
```

marc.poinot@onera.fr

Special methods - 3

- Context Manager

  `__enter__ __exit__`

  - Call from *with as* clause
  - Prepare a context and release/ clean context

- More attribute isolation

  `__getattr__ __setattr__`

  - Trap attribute access
  - Attribute should not already exist
  - On the fly check/ generation

marc.poinot@onera.fr

Reuse

- Derive from python objects
  - Base class of all classes
    - object
    - Recommanded in 2.x
  - Extensions types
    - ndarray
  - Exception classes
    - Exception

```
bool
int
float
str
unicode
tuple
list
dict
set
frozenset
map
```

Practical Training - 1

- Forewords
  - You can find simpler ways to define classes and methods and other mechanisms. The training tries to show many features in a short amount of time and in short pieces of software. The training classes are complex on purpose.

- Add classes in a new meshes.py
  - Change square, rectangle, cube, parallelepiped function into classes of same name
    - Use Mesh(ndarray) and Serialization(object) as base class
    - Add something like g0.dbg('Rectangle new') to print debug trace
    - Add something like Mesh.set_dbg(True) to set trace on
  - Create Exception for max size exceeded

- Add introspection functions
  - issubclass, isinstance, mro, inspect.methods...

marc.poinot@onera.fr

Practical Training - 2

- Serialization
  - Store the current state of the object
  - Use Pickle
  - Use a specific constructor
    - Hold all context/ state of object

- Write a generator to perform:

```
for m in Mesh.notArchivedinstances():
  m.doArchive()
```

marc.poinot@onera.fr

Practical Training - 3

- Build a factory

marc.poinot@onera.fr