

XVIII Escola de Verão IAG/USP

Funções e Objetos

Marcelo Bianchi
Victor Sacek
Leonardo Uieda



Jan/2016

Parte III

- Funções (def)
 - Funções anônimas (lambdas)
 - Operadores map(), filter() and reduce()
 - Geradores (yields)
- Classes & Instancias (class)
 - Derivando classes para estender sua funcionalidade !

Funções

- São segmentos de códigos que executam uma determinada tarefa criando um isolamento de código !
- Encapsulamento de tarefa !
- Não “guardam” informação

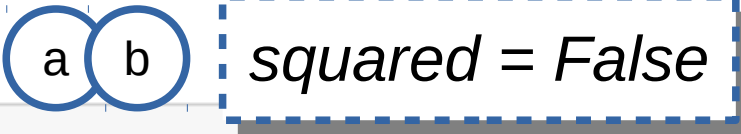
```
def soma(a, b, squared = False):  
    if squared:  
        a *= a  
        b *= b  
    return a + b
```

Chamando Funções em Python

```
def soma(a, b, squared = False):  
    if squared:  
        a *= a  
        b *= b  
    return a + b
```

Valor Padrão !

Implicito



`soma(1, 2)`

3

Explícito

`soma(a = 1, squared = True, b = 2)`

5

Lambda

- O operador lambda cria funções anônimas,
- Elas podem ser úteis para definir funções em tempo de execução

```
def makeshifter(n):  
    return lambda x: x + n
```

```
shift10 = makeshifter(10)  
shift20 = makeshifter(20)  
print type(shift10)
```

```
<type 'function'>
```

```
shift10(10)
```

```
20
```

```
shift20(10)
```

```
30
```

Map, filter & reduce

```
strlist = "1 2 3 4 5 6 7 8 9 10".split(" ")  
print strlist
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9',  
'10']
```

map(Método,
Iterável)

Aplica 1 à 1

filter(Método,
Iterável)

Filtra 1 à 1

reduce(Método,
Iterável)

Reduz 2 à 1

Map, filter & reduce

```
strlist = "1 2 3 4 5 6 7 8 9 10".split(" ")  
print strlist
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9',  
'10']
```

map(Método,
Iterável)

```
intlist = map(int, strlist)  
print intlist
```

Aplica 1 à 1

filter(Método,
Iterável)

Filtra 1 à 1

reduce(Método,
Iterável)

Reduz 2 à 1

Map, filter & reduce

```
strlist = "1 2 3 4 5 6 7 8 9 10".split(" ")  
print strlist
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9',  
'10']
```

map(Método,
Iterável)

```
intlist = map(int, strlist)  
print intlist
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Aplica 1 à 1

filter(Método,
Iterável)

```
odd = filter(lambda x: x % 2 == 0, intlist)  
print odd
```

Filtra 1 à 1

reduce(Método,
Iterável)

Reduz 2 à 1

Map, filter & reduce

```
strlist = "1 2 3 4 5 6 7 8 9 10".split(" ")  
print strlist
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9',  
'10']
```

map(Método,
Iterável)

```
intlist = map(int, strlist)  
print intlist
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Aplica 1 à 1

filter(Método,
Iterável)

```
odd = filter(lambda x: x % 2 == 0, intlist)  
print odd
```

```
[2, 4, 6, 8, 10]
```

Filtra 1 à 1

reduce(Método,
Iterável)

```
reduce(lambda x,y: x+y, odd)
```

Reduz 2 à 1

Map, filter & reduce

```
strlist = "1 2 3 4 5 6 7 8 9 10".split(" ")  
print strlist
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9',  
'10']
```

map(Método,
Iterável)

```
intlist = map(int, strlist)  
print intlist
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Aplica 1 à 1

filter(Método,
Iterável)

```
odd = filter(lambda x: x % 2 == 0, intlist)  
print odd
```

```
[2, 4, 6, 8, 10]
```

Filtra 1 à 1

reduce(Método,
Iterável)

```
reduce(lambda x,y: x+y, odd)
```

```
30
```

Reduz 2 à 1

Geradores

- São função que retornam parcialmente, podendo ser usadas no lugar de objetos iteráveis !
- A construção mais simples de geradores é a partir do uso da palavra ***yield***

```
def nsquare(f, t = None):  
    while t == None or f < t:  
        yield f**2  
        f += 1
```

- Função geradoras quando chamadas retornar geradores iteráveis e não o valor da função !!

```
generator = nsquare(10, 12)
print type(generator)
```

```
<type 'generator'>
```

- Para obter valores é necessário usar o comando **next()** ou utilizar o laço **for** !
- Quando a função chega ao fim (return), ela interrompe o laço ao emitir uma exceção do tipo **StopExecution**

```
print "0 próximo valor é: %f" % next(generator)
for f in generator:
    print f
```

```
0 próximo valor é: 100.000000
121
```

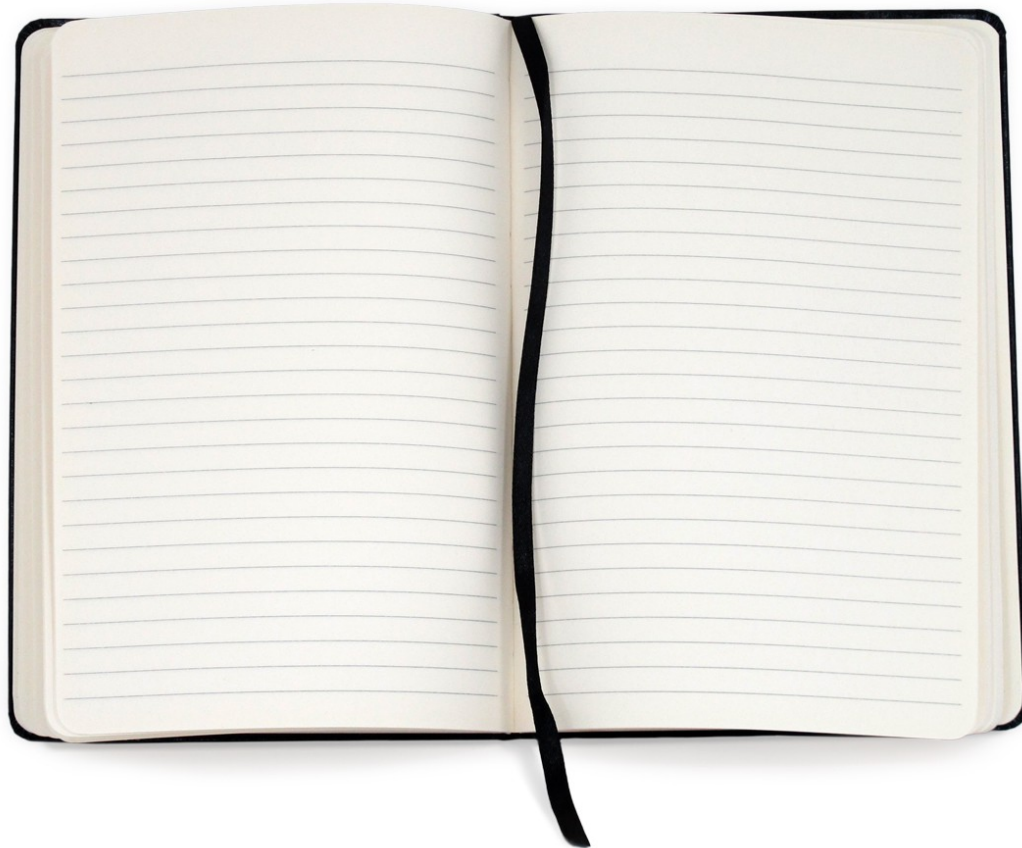
```
print "0 próximo valor é: %f" % next(generator)
```

```
-----
-----
StopIteration                                     Traceback
ack (most recent call last)
<ipython-input-176-6d9b8870656b> in <module>()
----> 1 print "0 próximo valor é: %f" % next(generator)
```

```
StopIteration:
```

Prática

- Trabalhe no notebook, 05-FuncaoGeradores



Abstraindo + !

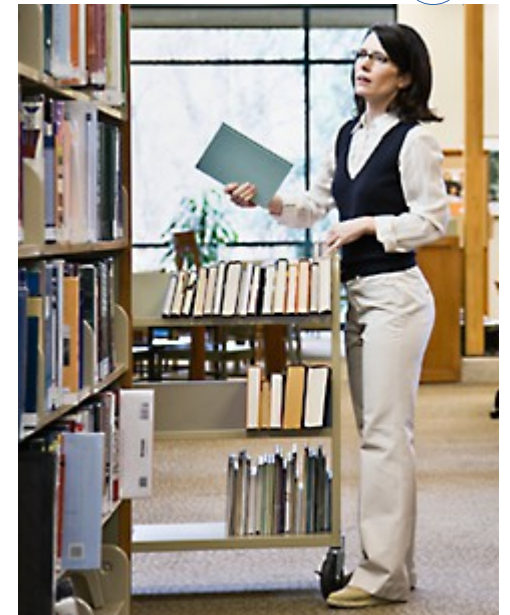
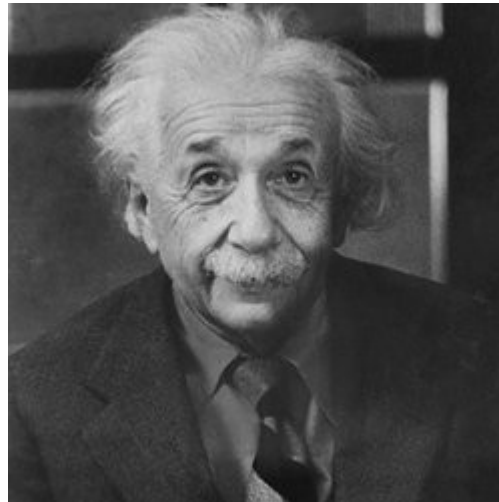


Classe

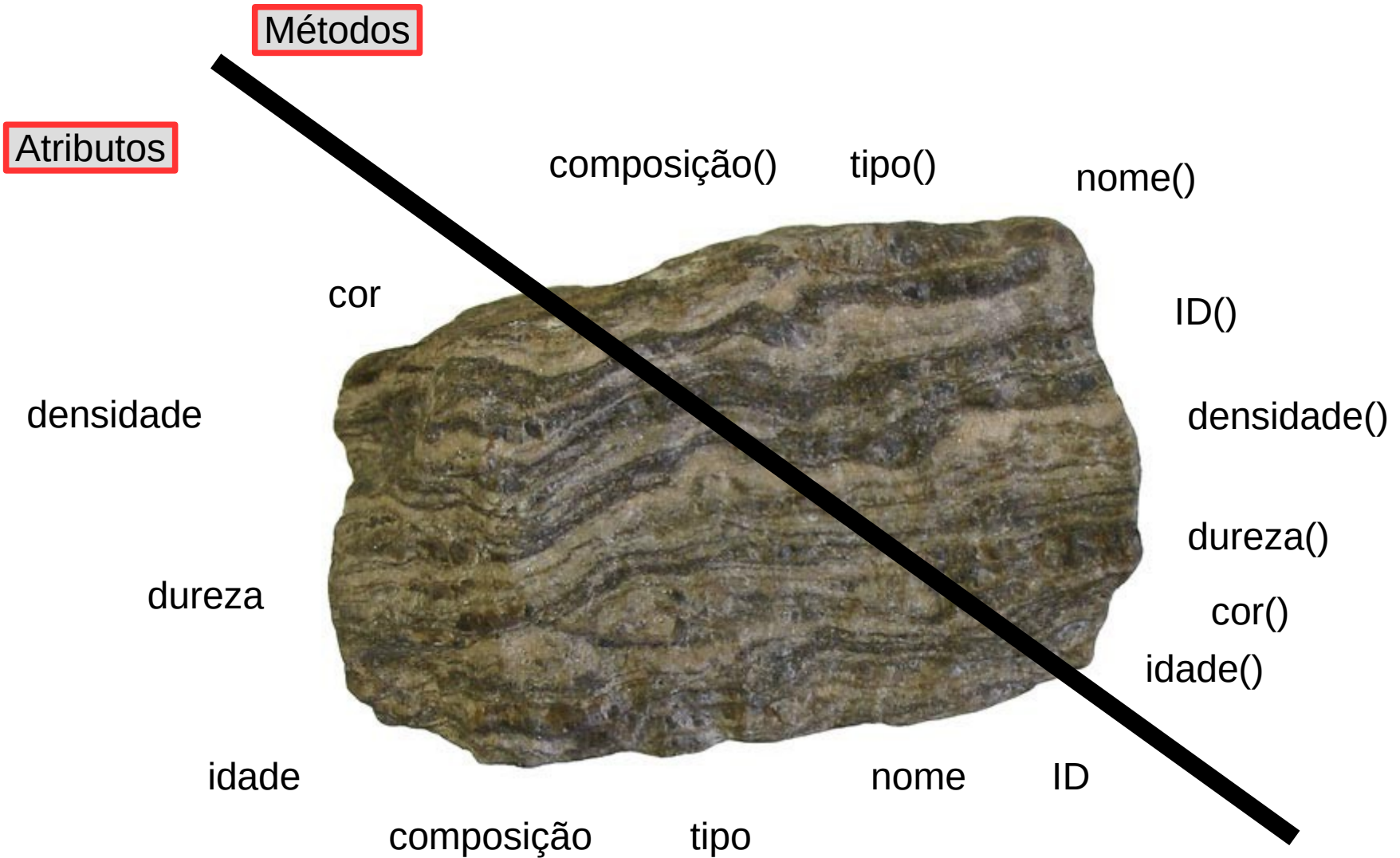
- Conceitos associados:
 - Atributos
 - Métodos
 - Instância

Classe

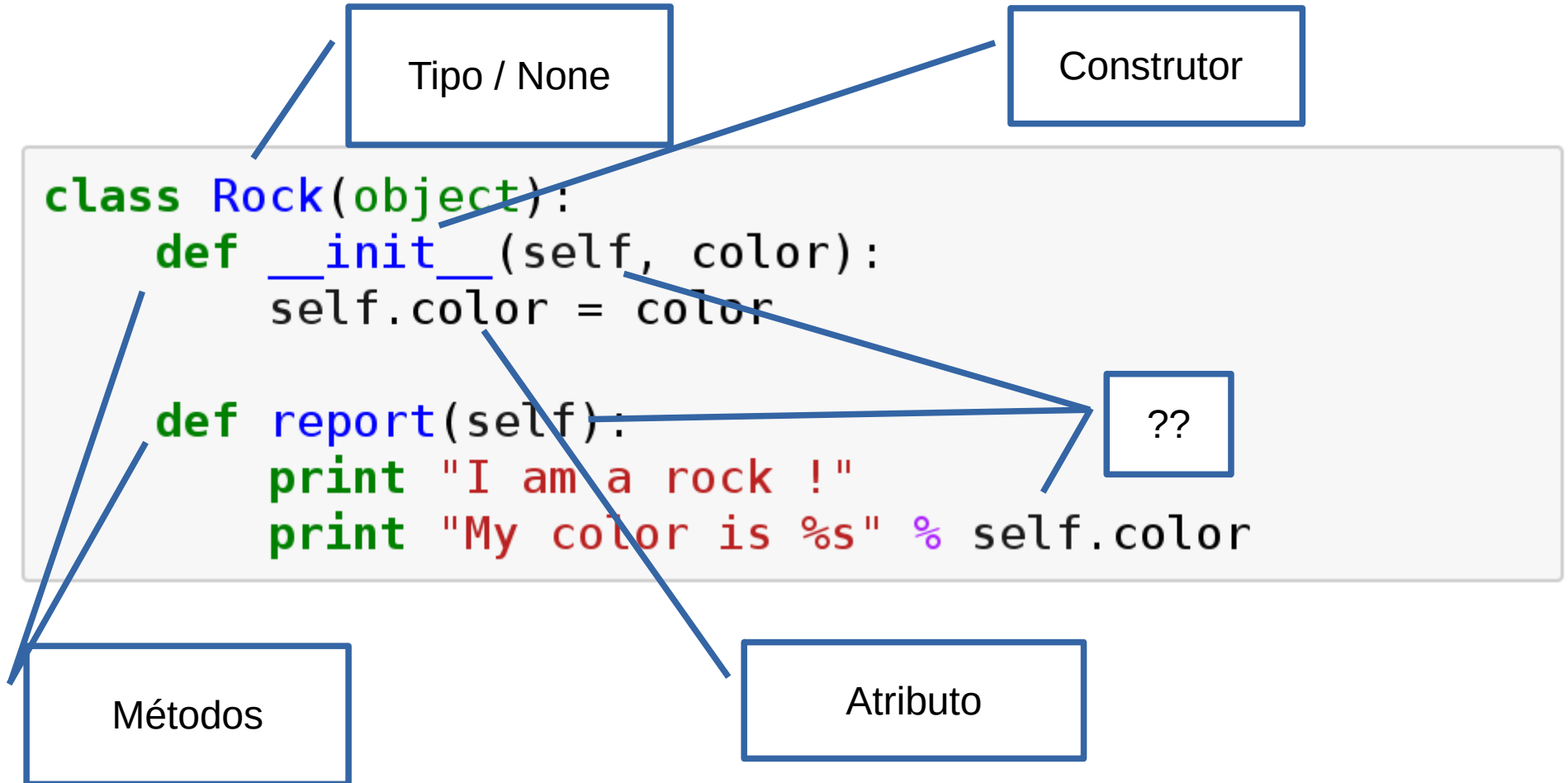
- Conceitos associados:
 - Atributos
 - Métodos
 - Instância



Atributos & Métodos



Definindo uma Classe



Classe



Instância I

?



Instância II

Criando uma Instância

```
class Rock(object):  
    def __init__(self, color):  
        self.color = color  
  
    def report(self):  
        print "I am a rock !"  
        print "My color is %s" % self.color
```

```
basalt = Rock("black")  
basalt.report()
```

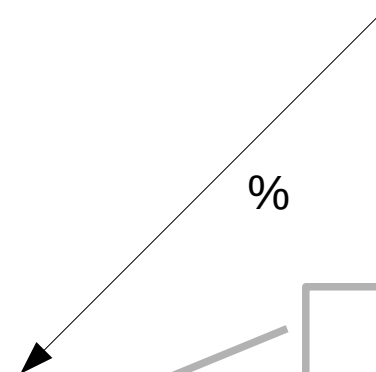
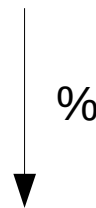
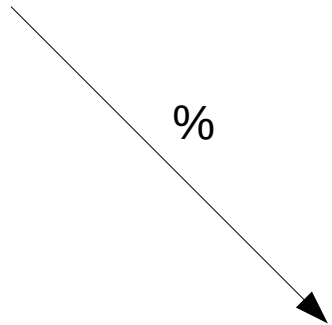
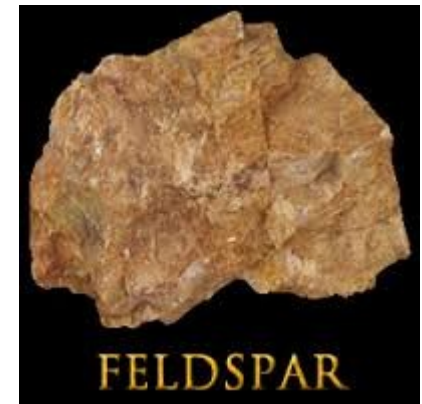
```
I am a rock !  
My color is black
```

E se a cor pode-se ser “computada”

- Ou seja: ser o resultado de uma operação pré-definida, normalmente baseada em outras “propriedades”



E se a cor pode-se ser “computada”



No caso +
simples a cor
poderia ser a
cor do mineral
predominante !

```
class Mineral(object):
    def __init__(self, name, color):
        self.name = name
        self.color = color

class Rock(object):
    def __init__(self):
        self.minerals = { }

    def addMineral(self, mineral, perc):
        if mineral.name in self.minerals:
            raise Exception("Error !")
        self.minerals[mineral.name] = (mineral, perc)

    def color(self):
        color = None
        maxp = 0
        for m,p in self.minerals.itervalues():
            if p > maxp:
                color = m.color
                maxp = p
        return color

    def report(self):
        print "I am a rock !"
        print "I have %d minerals, to know: %s" % (len(self.minerals), self.minerals.keys())
        print "My color is %s" % self.color()
```

```
nr = Rock()  
nr.addMineral(Mineral("quartz", "white"), 80)  
nr.addMineral(Mineral("mica", "brightyellow"), 10)  
nr.addMineral(Mineral("feldspar", "yellow"), 10)
```

```
nr.report()
```

```
I am a rock !  
I have 3 minerals, to know: ['quartz', 'mica', 'feldspar']  
My color is white
```



```

class Mineral(object):
    def __init__(self, name, color):
        self.name = name
        self.color = color

class Rock(object):
    def __init__(self):
        self.minerals = { }

    def addMineral(self, mineral, perc):
        if mineral.name in self.minerals:
            raise Exception("Error !")
        self.minerals[mineral.name] = (mineral, perc)

    def color(self):
        color = None
        maxp = 0
        for m,p in self.minerals.itervalues():
            if p > maxp:
                color = m.color
                maxp = p
        return color

    def report(self):
        print "I am a rock !"
        print "I have %d minerals, to know: %s" % (len(self.minerals), self.minerals.keys())
        print "My color is %s" % self.color()

```

```

nr = Rock()
nr.addMineral(Mineral("quartz", "white"), 80)
nr.addMineral(Mineral("mica", "brightyellow"), 10)
nr.addMineral(Mineral("feldspar", "yellow"), 10)

```

```
nr.report()
```

```

I am a rock !
I have 3 minerals, to know: ['quartz', 'mica', 'feldspar']
My color is white

```

Atributos Ocultos / Privados

```
class Rock(object):  
    def __init__(self, color, name):  
        self.color = color  
        self._name = name  
  
    def report(self):  
        print "I am a Rock: %s" % self._name  
        print "I am %s" % self.color
```

- `__*` Atributos Internos - evite !
- `__` Atributos forçadamente (name mangling) inacessíveis !!
- `_` Atributos ocultos mas não inacessíveis !!!