

Luciano Ramalho
luciano@ramalho.org

@pythonprobr



novembro/2013

Objetos Pythonicos

Orientação a objetos e padrões de projeto em Python

Aula 5

- Polimorfismo
- Metaprogramação e o Python Data Model
- Sobrecarga de operadores
- Objetos invocáveis
- Injeção de dependência

Polimorfismo: definição

O conceito de “polimorfismo” significa que podemos tratar instâncias de diferentes classes da mesma forma.

Assim, podemos enviar uma mensagem a um objeto sem saber de antemão qual é o seu tipo, e o objeto ainda assim fará “a coisa certa”, ao menos do ponto de vista dele.

Scott Ambler

The Object Primer, 2nd ed. - p. 173

Exemplos de polimorfismo

- A função dobro e o operador *
- A classe Baralho como sequência mutável
- live-coding com monkey-patching
 - programação ao vivo com modificação de classe em tempo de execução

Baralho polimórfico



Carta de baralho

```
class Carta(object):

    naipes = 'paus ouros copas espadas'.split()
    valores = '2 3 4 5 6 7 8 9 10 J Q K A'.split()

    def __init__(self, valor, naipe):
        self.valor = valor
        self.naipe = naipe

    def __repr__(self):
        return 'Carta(%r, %r)' % (self.valor, self.naipe)

    def __str__(self):
        return self.valor + ' de ' + self.naipe

    @classmethod
    def todas(cls):
        return [cls(v, n) for n in cls.naipes
                for v in cls.valores]
```


Carta de baralho

```
class Carta(object):

    naipes = 'paus ouros copas espadas'.split()
    valores = '2 3 4 5 6 7 8 9 10 J Q K A'.split()

    def __init__(self, valor, naipe):
        self.valor = valor
        self.naipe = naipe

    def __repr__(self):
        return 'Carta(%r, %r)' % (self.valor, self.naipe)

    def __str__(self):
        return self.valor + ' de ' + self.naipe

    @classmethod
    def todas(cls):
        return [cls(v, n) for n in cls.naipes
                for v in cls.valores]
```

```
>>> zape = Carta('4',
'paus')
>>> zape.valor
'4'
>>> zape
Carta('4', 'paus')
>>> monte = Carta.todas()
>>> len(monte)
52
>>> monte[0]
Carta('2', 'espadas')
>>> monte[-3:]
[Carta('Q', 'copas'),
Carta('K', 'copas'),
Carta('A', 'copas')]
```

Baralho

polimórfico (demo)

```
from carta_ord import Carta

class Baralho(object):

    def __init__(self):
        self.cartas = Carta.todas()

    def __len__(self):
        return len(self.cartas)

    def __getitem__(self, pos):
        return self.cartas[pos]
```


Baralho

polimórfico (final)

```
from carta_ord import Carta

class Baralho(object):

    def __init__(self):
        self.cartas = Carta.todas()

    def __len__(self):
        return len(self.cartas)

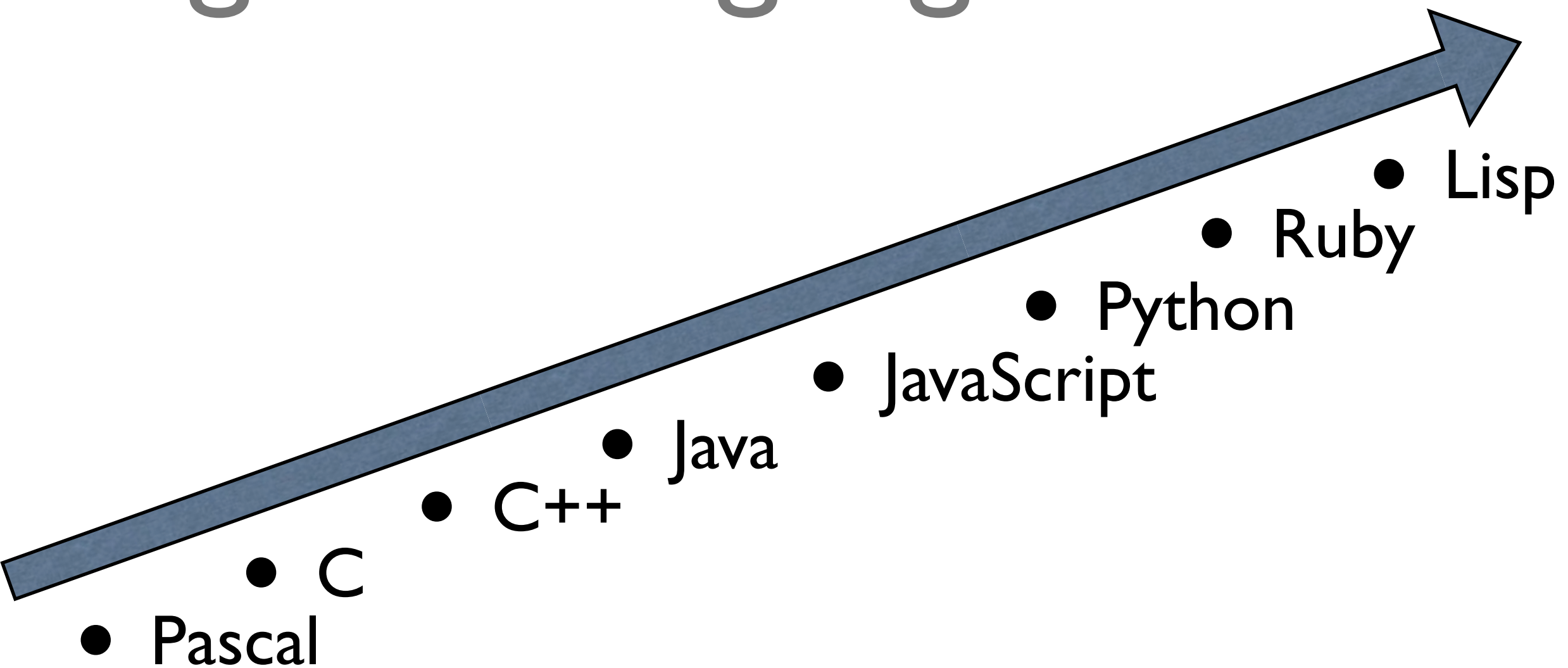
    def __getitem__(self, pos):
        return self.cartas[pos]

    def __setitem__(self, pos, valor):
        self.cartas[pos] = valor
```

Metaprogramação

- Escrever metaprogramas
 - programas que criam ou modificam programas
- Metaprogramas estáticos:
 - compiladores, interpretadores, pre-processadores
- Metaprogramas dinâmicos
 - interpretadores, frameworks, DSLs etc.

Potencial de metaprogramação em algumas linguagens

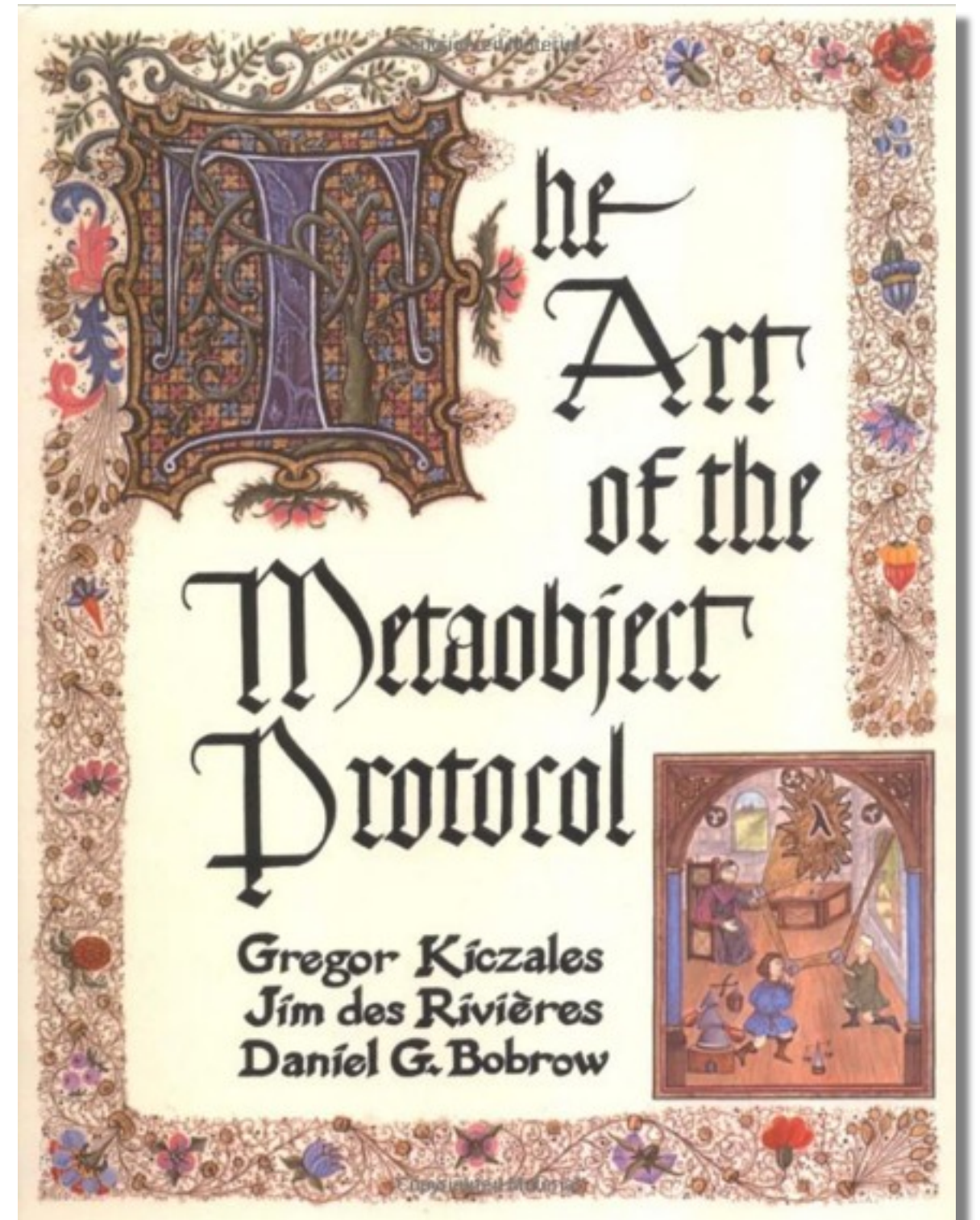


Metaprogramação Orientada a Objetos

- MOP: Meta Object Protocol
- API para a manipulação de objetos em tempo de execução
- os objetos manipulados podem ser **classes**, **funções**, **closures**, **módulos**, **bytecode** e outros objetos que nas linguagens estáticas só podem ser construídos em tempo de compilação

Livro clássico: A.M.O.P.

- The Art of the Metaobject Protocol
- CLOS: Common Lisp Object System
- para entender Ruby, Python e JavaScript a fundo



Exemplos de MOPs

- Algumas linguagens com protocolos de metaobjetos
 - Java, JavaScript, PHP, Python, Ruby, Scheme, Lisp...
- Exemplo concreto:
 - Python Data Model: o MOP da linguagem Python



Python

Data

Model

dunder?



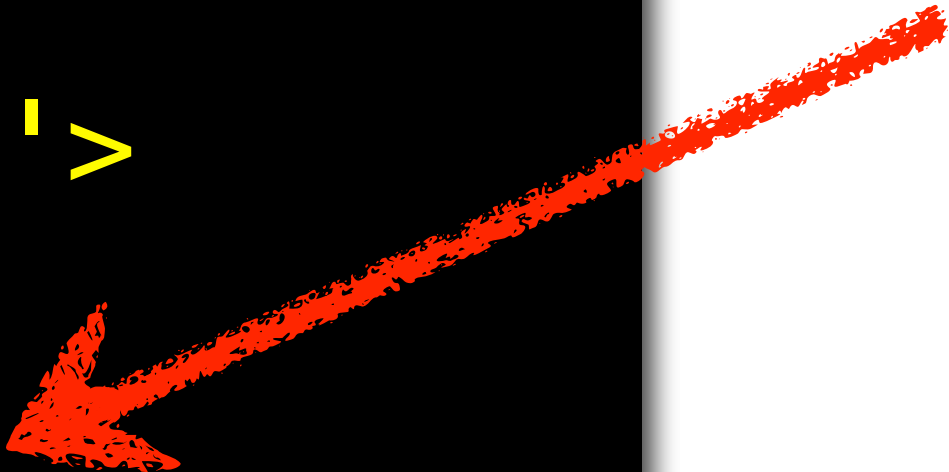
The background of the image is a dramatic sky filled with dark, swirling clouds in shades of deep blue and purple. Several bright, jagged yellow lightning bolts are visible, striking downwards from the upper part of the frame. The word "thunder" is written in a bold, yellow, sans-serif font across the lower center of the image.

thunder

The image shows a dramatic sky with dark blue and purple clouds. Several bright yellow lightning bolts are visible, striking downwards from the clouds. The word "thunder" is written in a bold, yellow, sans-serif font across the lower part of the image.

thunder

```
>>> x = 7
>>> type(x)
<class 'int'>
>>> x * 6
42
>>> x.__mul__(6)
42
```

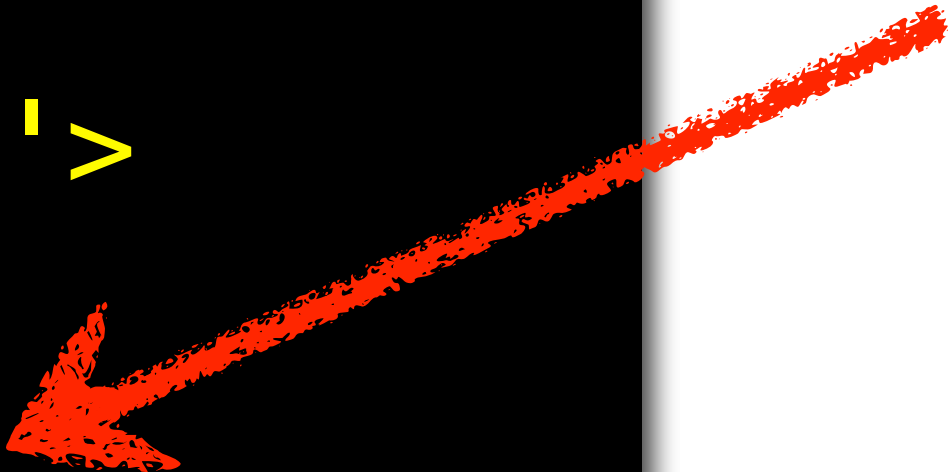


double underscore

mul

double underscore

```
>>> x = 7
>>> type(x)
<class 'int'>
>>> x * 6
42
>>> x.__mul__(6)
42
```



dunder mul!

atributos de um int

```
>>> dir(7)
['__abs__', '__add__', '__and__', '__bool__',
 '__ceil__', '__class__', '__delattr__', '__divmod__',
 '__doc__', '__eq__', '__float__', '__floor__',
 '__floordiv__', '__format__', '__ge__',
 '__getattr__', '__getnewargs__', '__gt__',
 '__hash__', '__index__', '__init__', '__int__',
 '__invert__', '__le__', '__lshift__', '__lt__',
 '__mod__', '__mul__', '__ne__', '__neg__', '__new__',
 '__or__', '__pos__', '__pow__', '__radd__', '__rand__',
 '__rdivmod__', '__reduce__', '__reduce_ex__',
 '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__',
 '__rmul__', '__ror__', '__round__', '__rpow__',
 '__rrshift__', '__rshift__', '__rsub__',
 '__rtruediv__', '__rxor__', '__setattr__',
 '__sizeof__', '__str__', '__sub__', '__subclasshook__',
 '__truediv__', '__trunc__', '__xor__', 'bit_length',
 'conjugate', 'denominator', 'from_bytes', 'imag',
 'numerator', 'real', 'to_bytes']
```


atributos de uma str

```
>>> dir('abc')
['__add__', '__class__', '__contains__', '__delattr__',
 '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__getnewargs__',
 '__gt__', '__hash__', '__init__', '__iter__', '__le__',
 '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
 '__new__', '__reduce__', '__reduce_ex__', '__repr__',
 '__rmod__', '__rmul__', '__setattr__', '__sizeof__',
 '__str__', '__subclasshook__', 'capitalize', 'center',
 'count', 'encode', 'endswith', 'expandtabs', 'find',
 'format', 'format_map', 'index', 'isalnum', 'isalpha',
 'isdecimal', 'isdigit', 'isidentifier', 'islower',
 'isnumeric', 'isprintable', 'isspace', 'istitle',
 'isupper', 'join', 'ljust', 'lower', 'lstrip',
 'maketrans', 'partition', 'replace', 'rfind', 'rindex',
 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
 'splitlines', 'startswith', 'strip', 'swapcase',
 'title', 'translate', 'upper', 'zfill']
```

atributos comuns a int e str

```
>>> sorted(set(dir(7)) & set(dir('abc')))  
['__add__', '__class__', '__delattr__', '__doc__',  
 '__eq__', '__format__', '__ge__', '__getattribute__',  
 '__getnewargs__', '__gt__', '__hash__', '__init__',  
 '__le__', '__lt__', '__mod__', '__mul__', '__ne__',  
 '__new__', '__reduce__', '__reduce_ex__', '__repr__',  
 '__rmod__', '__rmul__', '__setattr__', '__sizeof__',  
 '__str__', '__subclasshook__']
```

atributos comuns a str e list

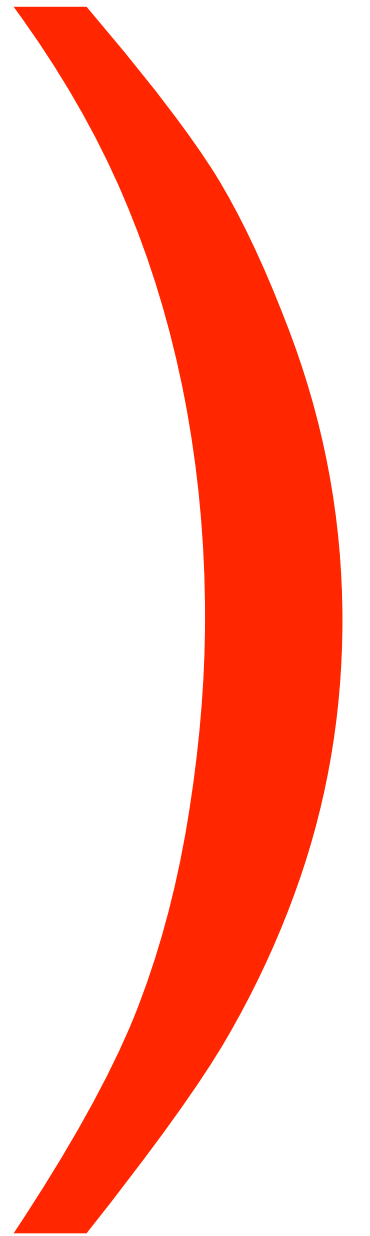
```
>>> sorted(set(dir('abc')) & set(dir([])))  
['__add__', '__class__', '__contains__', '__delattr__',  
 '__doc__', '__eq__', '__format__', '__ge__',  
 '__getattribute__', '__getitem__', '__gt__',  
 '__hash__', '__init__', '__iter__', '__le__',  
 '__len__', '__lt__', '__mul__', '__ne__', '__new__',  
 '__reduce__', '__reduce_ex__', '__repr__', '__rmul__',  
 '__setattr__', '__sizeof__', '__str__',  
 '__subclasshook__', 'count', 'index']
```

Métodos dunder

= “special methods”

- Protocolos genéricos, quase universais
- Úteis para muitas classes em muitos contextos
- **O interpretador Python** invoca estes métodos em determinados contextos
- conversão, operadores, acesso a atributos e itens, iteração, ciclo de vida do objeto etc.

Python
Data Model:
special methods



Sobrecarga de operadores

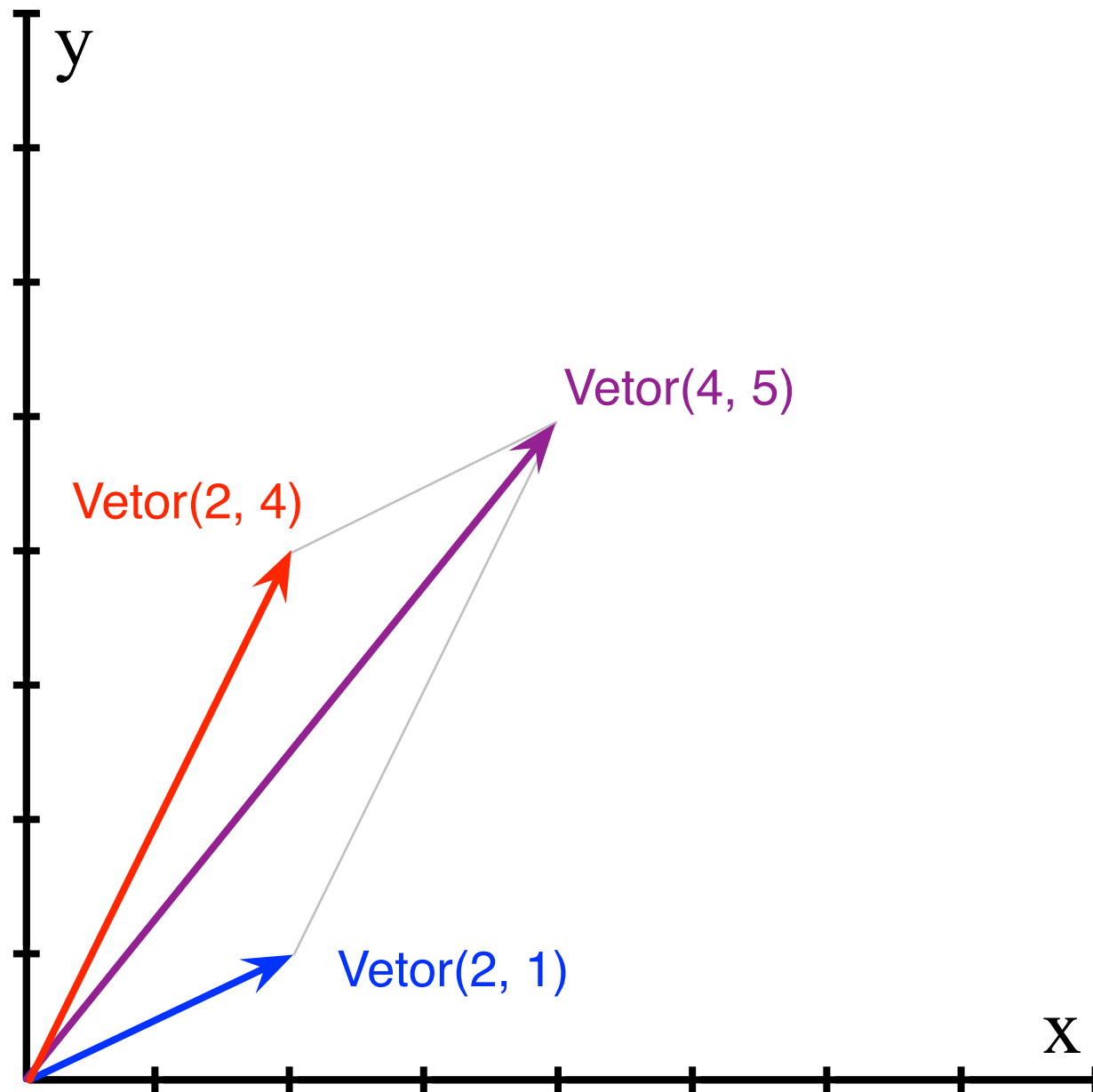
- Python permite que as classes definidas pelo usuário (você!) implementem métodos para os operadores definidos na linguagem
- Não é possível redefinir a função dos operadores nos tipos embutidos
 - isso evita surpresas desagradáveis
- Nem é possível inventar novos operadores
 - não podemos definir \sim , \leq , $/\backslash$ etc.

Alguns operadores existentes

- Aritméticos: + - * / ** //
- Bitwise: & ^ | << >>
- Acesso a atributos: a.b
- Invocação: f(x)
- Operações em coleções: c[a], len(c), a in c, iter(c)
- Lista completa em Python Reference: Data Model

<http://docs.python.org/reference/datamodel.html>

Exemplo: vetor (2d)



- Campos: x, y
- Métodos:
 - `distancia`
 - `abs` (distância até 0,0)
 - `+` (`__add__`)
 - `*` (`__mul__`) escalar



git

[oopy/exemplos/vetor.py](https://oopy.com.br/exemplos/vetor.py)

@pythonprobr

Vetor

```
from math import sqrt

class Vetor(object):

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __repr__(self):
        return 'Vetor(%s, %s)' % (self.x, self.y)

    def distancia(self, v2):
        dx = self.x - v2.x
        dy = self.y - v2.y
        return sqrt(dx*dx + dy*dy)

    def __abs__(self):
        return self.distancia(Vetor(0,0))

    def __add__(self, v2):
        dx = self.x + v2.x
        dy = self.y + v2.y
        return Vetor(dx, dy)

    def __mul__(self, n):
        return Vetor(self.x*n, self.y*n)
```

```
>>> from vetor import Vetor
>>> v = Vetor(3, 4)
>>> abs(v)
5.0
>>> v1 = Vetor(2, 4)
>>> v2 = Vetor(2, 1)
>>> v1 + v2
Vetor(4, 5)
>>> v1 * 3
Vetor(6, 12)
```

Objetos invocáveis

- Toda função é um objeto em Python
- Você pode definir suas próprias funções...
- E também novas classes de objetos que se comportam como funções: objetos invocáveis
 - basta definir um método `__call__` para sobrecarregar o operador de invocação: `o(x)`
- Exemplo: tómbola invocável

Tômbola invocável

- Já que o principal uso de uma instância de tômbola é sortear, podemos criar um atalho: em vez de **t.sortear()** apenas **t()**

```
from tombola import Tombola
```

```
class TombolaInvocavel(Tombola):  
    '''Sorteia itens sem repetir;  
    a instância é invocável como uma função'''
```

```
def __call__(self):  
    return self.sortear()
```

```
>>> t = TombolaInvocavel()  
>>> t.carregar([1, 2, 3])  
>>> t()  
3  
>>> t()  
2
```

Injeção de dependência

- Componente cliente requisita serviço de um componente provedor
- O componente provedor depende de um terceiro componente para realizar seu serviço (essa é a dependência)
- Na injeção de dependência, o cliente fornece ao provedor a tal dependência
- Exemplo simples: classe de data “congelada” para facilitar testes automatizados

Data congelada

- Para injeção em testes que dependem de **date.today()**, a classe **FrozenDate** devolve sempre a mesma data para “hoje”*

```
from datetime import date
```

```
class FrozenDate(date):
```

```
    @staticmethod
```

```
    def today():
```

```
        return date.fromtimestamp(10**9) # 2001-09-08
```

```
>>> d = FrozenDate.today()  
>>> d  
datetime.date(2001, 9, 8)
```

* 8/set/2001 foi o dia do segundo 1.000.000 da época Unix