Python

& Tradução de código para Common Lisp

Python

& Tradução de código para Common Lisp

Introdução ao Python

Object Oriented em Python

pnil

Maiores Problemas em traduzir Python para Common Lisp

Ducktype

Yield & With

Visitors & Dispatching

Decorators

Alteração de métodos

Python

Introdução

multi-paradigm: object-oriented, imperative, functional

strong, dynamic, duck

ABC, ALGOL 68,C, Haskell, Icon, Lisp, Modula-3, Perl, Java

CPython, Jython, IronPython, PyPy

Python

Curso Rapido de Python

```
def fact(n):
...if(n == 0):
......return 1
...else:
......return n * fib(n - 1)
```

```
>>> def f (x): return x**2
...
>>> print f(8)
64
>>>
>>> g = lambda x: x**2
>>>
>>> print g(8)
64
```

```
from xml.dom import minidom
import sys, time, urllib, simplejson
topic = sys.argv[1]
url = "http://search.twitter.com/search.json?q=%s&rpp=100" % topic
results = []
page = 1
nextpage = True
try:
    while nextpage:
        print "Page %d: %s" % (page,url)
        json = urllib.urlopen(url)

        doc = simplejson.load(json)

        nextpage = doc["next_page"]
        url = "http://search.twitter.com/search.json" + nextpage
        page = page + 1
        results.extend(doc["results"])
except KeyError:
    print "Got %d Results!" % len(results)
for result in results:
#for it in 0..len(results):
#result = results[it]
    print "%s : %s" % (result['from_user'],result['text'])
```

	Python	Common Lisp
Integer	`42`	`42`
Bignum	`1E+17`	`1E+17`
Float	`1,234`	`1,234`
Complex	`1 + 2J`	`#C(1, 2)`
String	`"hello" or 'hello'`	`"hello"`
Symbol	`'hello'`	`hello`
Hashtable/Dictionary	`{}`	`(make-hash-table)`
Function	`lambda x: x + x`	`(lambda (x) (+ x x))`
Class	`class Stack: ...`	`(defclass stack ...)`
Instance	`Stack()`	`(make-instance 'stack)`
Stream	`open("file")`	`(open "file")`
Boolean	`True, False`	`t, nil`
Empty Sequence	`(), [] tuple, array`	`(), #()`
Missing Value	`None`	`nil`
Lisp List (linked)	`(1, (2.0, ("three", None)))`	`(1 2.0 "three")`
Python List (adjustable array)	`[1, 2.0, "three"]`	`(make-array 3 :adjustable t ...`

Python

Object Oriented

```
class Queue(object):
    def __init__(self):
        self.__object = []

    def push(self,obj):
        self.__object.push(obj)

     def pop(self.obj):
   return self.__object.pop()

a = Queue()
a.push(1)
print a.pop()
```

```
foo.meth(arg) == C.meth(foo, arg)
```

pnil

pnil

Maiores Problemas

Inner Classes

Duck typing

Semantica não presente em Common Lisp (yield, with)

pnil

Inner Classes

Em Python é possível definir classes dentro de classes (e até de funções/métodos).

Em Common Lisp nem os métodos pertencem ás classes.

Python

Duck Typing

“Pythonic programming style that determines an object's type by inspection of its method or attribute signature rather than by explicit relationship to some type object ("If it looks like a duck and quacks like a duck, it must be a duck.") By emphasizing interfaces rather than specific types, well-designed code improves its flexibility by allowing polymorphic substitution. Duck-typing avoids tests using type() or isinstance(). Instead, it typically employs the EAFP (Easier to Ask Forgiveness than Permission) style of programming.”

```
if hasattr(mallard, "quack"):
    ...
```

```
try:
    mallard.quack()
except (AttributeError, TypeError):
    print >> sys.stderr "mallard can't quack()")
```

Python

Duck Typing

```
def foo (a,b):
    return a % b
```

```
>>> foo (1,2)
1
>>> foo (“hello %d!”, “jenny”)
'hello jenny!'
```

Python

Duck Typing

```
(shadow 'mod)
(defgeneric mod (a b))
(defmethod mod ((a integer) (b integer))
    (cl:mod a b))
(defmethod mod ((a string) (b string))
    (cl:format nil a b))
```

```
CL-USER> (mod 1 2)
1
CL-USER> (mod "hello ~A!" "jenny")
"hello jenny!"
```

Python

Generators

Principio semelhante às continuações de Scheme:

```
>>> def foo ():
...     print "a"
...     yield
...     print "b"
...     yield
...
>>>
```

```
>>> a = foo()
>>> a.next()
a
>>> a.next()
b
>>> a.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

Python

Generators

O Common Lisp não possui continuations!

A tradução é bastante difícil...

```
def fib():
    a, b = 0, 1
    while 1:
        yield b
        a, b = b, a+b
```

... mas possível!

Python

With

```
with EXPR as VAR:
    BLOCK
```

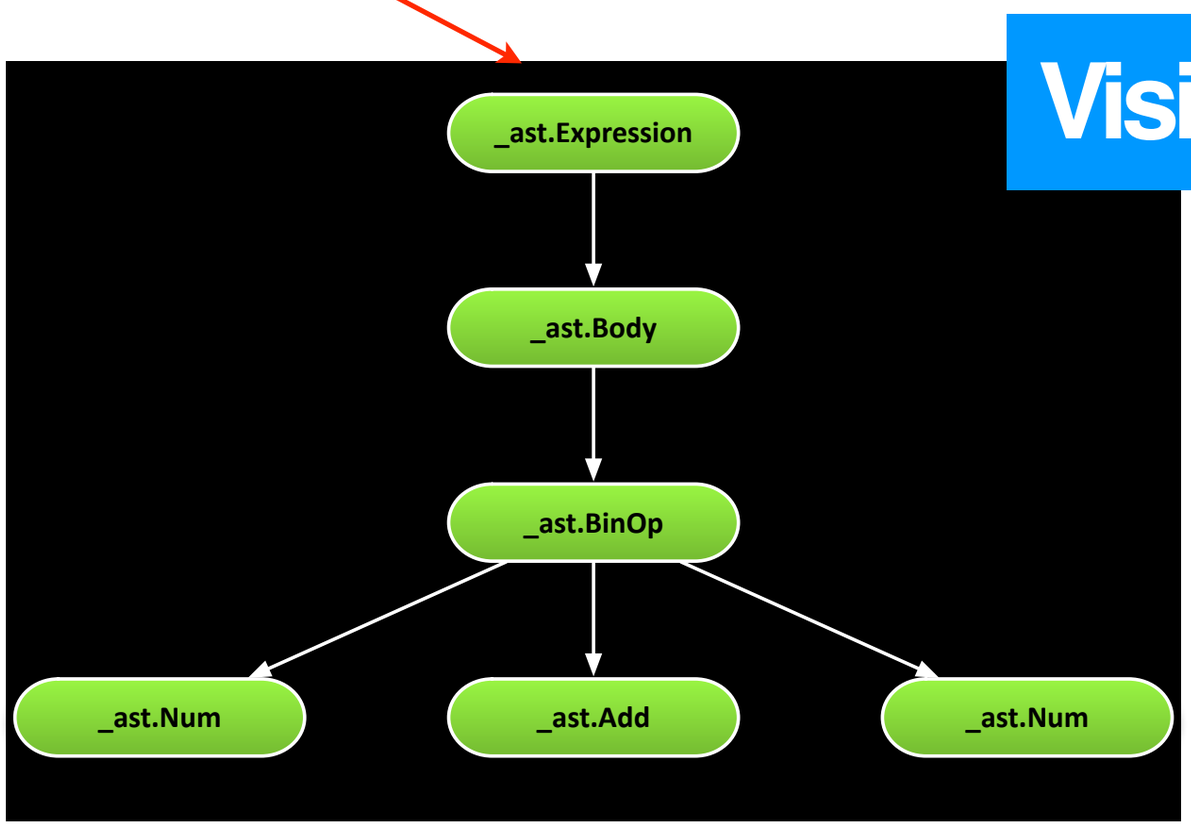
```
mgr = (EXPR)
exit = mgr.__exit__
value = mgr.__enter__()
exc = True
try:
    try:
        VAR = value
        BLOCK
    except:
        exc = False
        if not exit(*sys.exc_info()):
            raise
finally:
    if exc:
        exit(None, None, None)
```

pnil

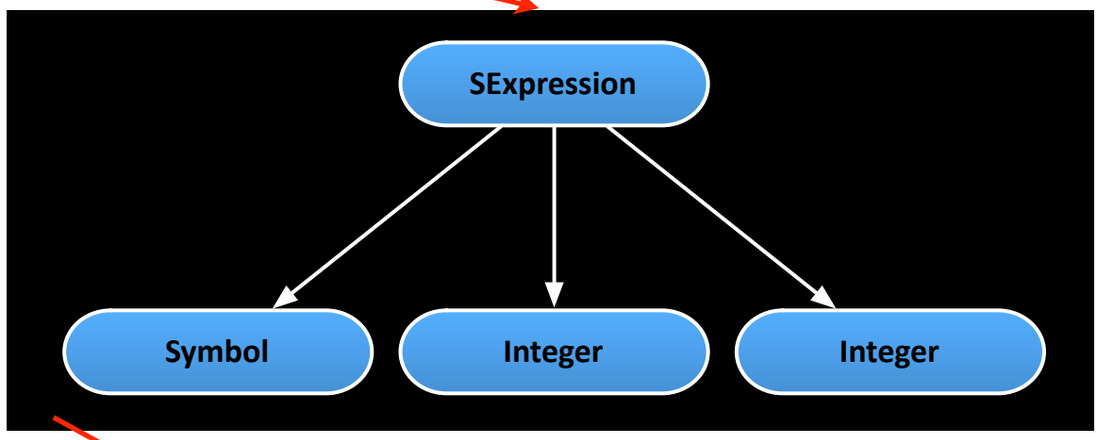
Internals

1+2

```
ast = compile(contents,argv[1],"exec",_ast.PyCF_ONLY_AST)
```



Visitor



(+12)

pnil

Internals

Visitor

Python não consegue decidir como despachar a mensagem devido ao duck typing

Java

Visitor

```
interface Visitor {
    void visit(Wheel wheel)
    void visit(Engine engine)
    ...
}
interface CarElement{
    void accept(Visitor visitor)
}
class Wheel implements CarElement{
    private String name
    Wheel(String name) {
        this.name = name;
    }
    String getName() {
        return this.name;
    }
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }
}
class Car  {
    CarElement[] elements;
    public CarElement [] getElements(){
        return elements clone();
    }
}
```

```
class CarElementPrintVisitor implements
CarElementVisitor {

    public void visit(Wheel wheel) {
        System.out.println("Visiting "+ wheel.getName
()
                                + " wheel");
    }
    public void visitCar(Car car) {
        System.out.println("\nVisiting car")
        for(CarElement element : car.getElements()) {
            element.accept(this);
        }
        System.out.println("Visited car");
    }
}
```

Common Lisp

Visitor

```
(defclass car-element ())

(defclass wheel (car-element)
((name (:type string
        :initarg name
        :initform “”
        :acessor wheel-name))))

(defclass engine (car-element))

(defclass car (car-element)
((elements (:type (list car-element)
            :initargs ()
            :accessor car-element)))
```

```
(defgeneric visit (obj))

(defmethod visit ((obj wheel))
   (format t “I’m visiting a wheel”))

(defmethod visit ((obj engine))
   (format t “I’m visiting a engine”))

(defmethod visit ((obj car))
   (format t “I’m visiting a car”)
   (loop for i in car-element do
      (visit (car-element obj))))
```

Python

Visitor

```
class CarElement (object):

class Wheel (CarElement):
   def __init__(name):
      self.name = name

class Engine (CarElement):

class Car (CarElement):
   def __init__():
      self.elements =[]
```

```
import dispatch

class Visitor (object):
    @dispatch.on('node')
    def visit(self, node):
        "Generic Method"

    @visit.when(Wheel)
    def visit(self,node):
       print “I’m visiting a Wheel”

    @visit.when(Engine)
    def visit(self,node):
       print “I’m visiting a Engine”

    @visit.when(Car)
    def visit(self,node):
       print “I’m visiting a Car”
       for i in node.elements:
          visit(node)
```

Python

Decorators

```
@classmethod
def foo(cls):
    perform method operation
```

```
def foo(self):
    perform method operation
foo = classmethod(foo)
```

```
class PostArticle(RequestHandler):
    @require_login
    def get(self):
        self.render("post.html")
```

Python

Alterar Métodos em runtime

```
>>> class FooBar (object):
...     def foo (self):
...         print "Foo!"
...
>>> a = FooBar()
>>> def foo (self):
...     print "Bar!"
...
>>> FooBar.foo = foo
>>> a.foo()
Bar!
>>>
```

Python 3000

Python 3000

Lançado ontem á noite

Quebra compatibilidade com o Python 2.x

Suporta código escrito em unicode

Alguma alterações relacionadas com o statement print

Fim

Questões?