PyLisp

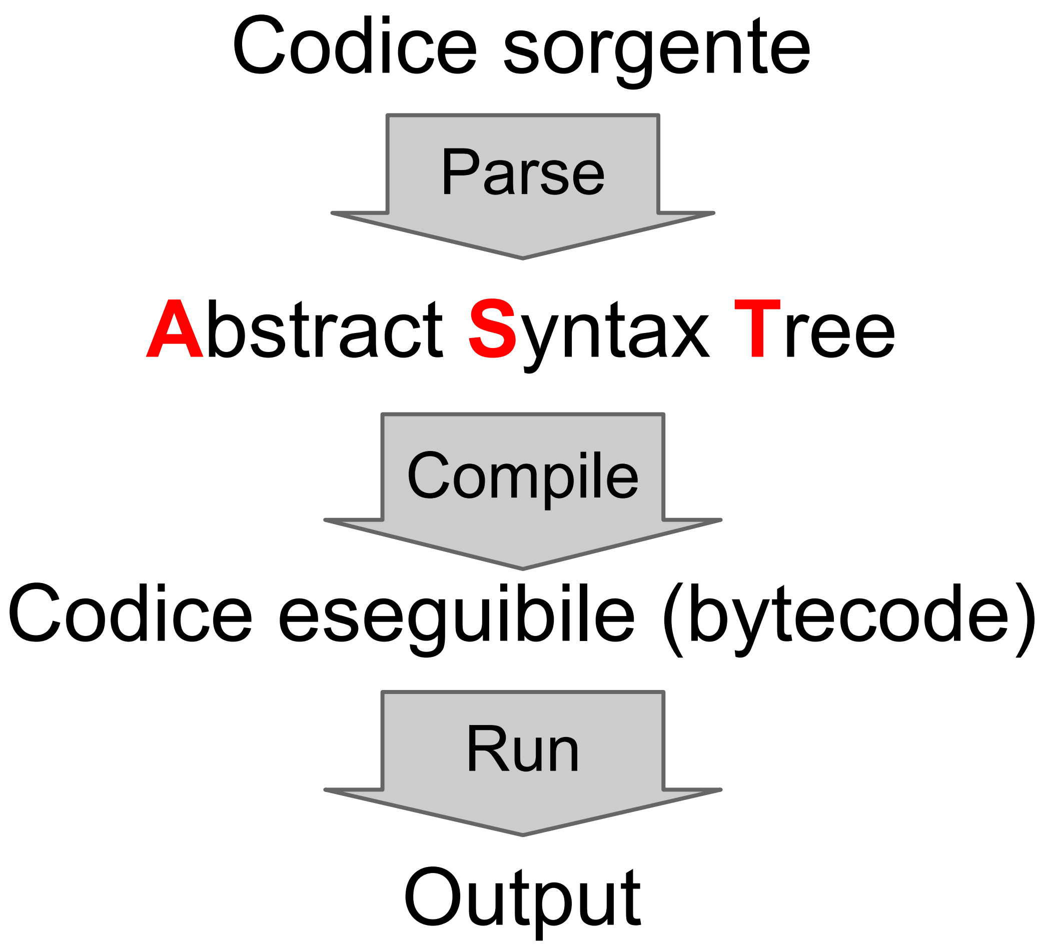
Compilare il LISP in Python bytecode

PyLisp

https://github.com/6502/pylisp

Andrea “6502” Griffini
agriff@tin.it
@agriffini

Funzionamento cpython



Esempio di bytecode

```
>>> def square(x):
...     return x * x
...
>>> import dis
>>> dis.dis(square)
  2           0 LOAD_FAST                0 (x)
              3 LOAD_FAST                0 (x)
              6 BINARY_MULTIPLY
              7 RETURN_VALUE
>>>
```

Esempio di bytecode (2)

```
>>> def fact(x):
...     if x < 2:
...         return 1
...     else:
...         return x * fact(x - 1)
...
>>>
```

```
2           0 LOAD_FAST                0 (x)
            3 LOAD_CONST               1 (2)
            6 COMPARE_OP               0 (<)
            9 POP_JUMP_IF_FALSE       16
3          12 LOAD_CONST               2 (1)
           15 RETURN_VALUE
5     >>   16 LOAD_FAST                0 (x)
           19 LOAD_GLOBAL              0 (fact)
           22 LOAD_FAST                0 (x)
           25 LOAD_CONST               2 (1)
           28 BINARY_SUBTRACT
           29 CALL_FUNCTION            1
           32 BINARY_MULTIPLY
           33 RETURN_VALUE
           34 LOAD_CONST               0 (None)
           37 RETURN_VALUE
```

L’oggetto __code__

co_filename	co_cellvars
co_firstlineno	co_names
co_lnotab	co_consts
co_name	co_varnames
co_nlocals	co_freevars
co_stacksize	co_code
co_argcount	co_flags

Problema “funarg”

Caso 1 (semplice: “downward funarg”)

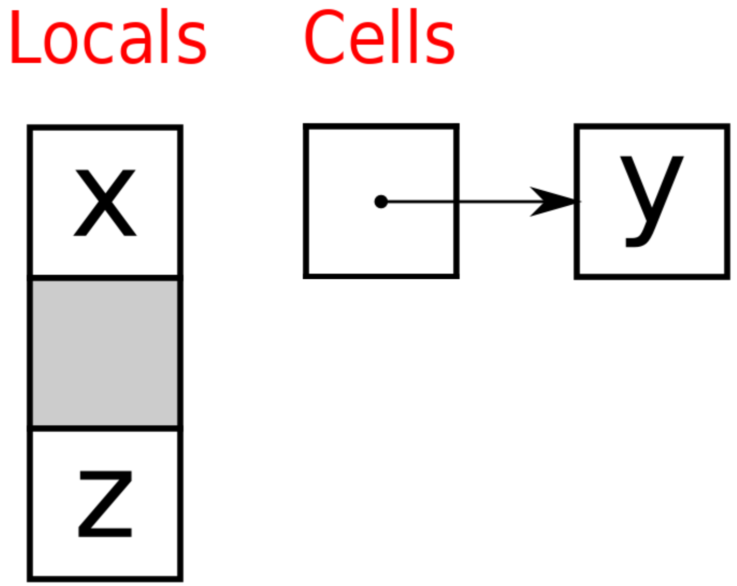
```
x = 12
L = map((lambda y: x * y),
        range(10))
```

Caso 2 (complesso: “upward funarg”)

```
def adder(x):
    return lambda y: x + y
```

Cosa sono cellvars e freevars

```
def foo(x, y, z):
    return [x, y+z, lambda n: n*y]
```



Cosa sono cellvars e freevars

```
>>> def adder(x):
...     return lambda y: x + y

>>> adder.__code__.co_cellvars
('x',)

>>> adder(42).__code__.co_freevars
('x',)
```

Cosa sono cellvars e freevars

```
>>> def adder(x):
...     return lambda y: x + y

>>> adder.__code__.co_cellvars
('x',)

>>> adder(42).__code__.co_freevars
('x',)
```

```python
def foo(x, y, z):
    return [x, y+z]
```

```
2     0 LOAD_FAST      0 (x)
      3 LOAD_FAST      1 (y)
      6 LOAD_FAST      2 (z)
      9 BINARY_ADD
     10 BUILD_LIST     2
     13 RETURN_VALUE
```

```
def foo(x, y, z):
    return [x, y+z, lambda n: n*y]


2     0 LOAD_FAST        0 (x)
      3 LOAD_DEREF       0 (y)
      6 LOAD_FAST        2 (z)
      9 BINARY_ADD
     10 LOAD_CLOSURE     0 (y)
     13 BUILD_TUPLE      1
     16 LOAD_CONST       1 (<code object ...>)
     19 MAKE_CLOSURE     0
     22 BUILD_LIST       3
     25 RETURN_VALUE
```

Python e Lisp

Similitudini

- Imperativi (multiparadigma)
- Tipizzazione dinamica
- Gestione memoria automatica
- Compilati a runtime, REPL

Differenze

- Sintassi
- Dinamicita’
- Metaprogrammazione
- Livello

La “sintassi” Lisp

Atomi

```
1 "bar" 42
```

Liste

```
(1 2 2 "bar")
(1 (2 3) 4)
```

Codice

```
(foo x 42 "bar" (+ z 3))
```

Lisp-1 e Lisp-2

```
(defun foo (foo)
  (if (< foo 2)
      1
      (* foo (foo (- foo 1)))))
```

Lisp-1 e Lisp-2

```
(defun foo (foo)
  (if (< foo 2)
      1
      (* foo (foo (- foo 1)))))
```

“Lisp has no syntax”

$$\begin{tikzcd}
\textbf{Testo}\text{ sorgente} \arrow[d, "\text{Read}"] \\
\textbf{Forms} \arrow[d, "\text{Compile}"] \\
\text{Codice eseguibile (bytecode)} \arrow[d, "\text{Run}"] \\
\text{Output}
\end{tikzcd}$$

“Lisp has no syntax”

In realta’ Lisp ha DUE sintassi

- La sintassi di reading
- La sintassi di compilazione

Ma nessuna delle due e’ immutabile

- Reader macros
- Macros

Esempio di sessione

```
~/checkout/pylisp/src$ python pylisp.py
PyLisp 0.006
> (defun square (x)
    (* x x))
--> <function lambda at 0x258e7d0>
> (square 12)
--> 144
> (dis #'square)
  0   0 LOAD_FAST           0 (_Lx)
      3 LOAD_FAST           0 (_Lx)
      6 BINARY_MULTIPLY
      7 RETURN_VALUE
```

pylisp.py

- Compatibilita’ Python 2/3
- Opcode generici load/store, labels
- Mangling
- Simboli
- Il compiler context `make_code`
- Primitive: `emit`, `bytecode`, `stackeffect`, `setq`, `progn`, `if`, `quote`, `function`, `lambda`, `fsetq`, `msetq`, `defun`, `defmacro`
- Runtime base: `python`, `eval`, `mapn`, `mapl`

pylisp.lisp

- Runtime esteso
- `dotimes`, `dolist`, `when`, `not`
- `lassoc-binop`
- `aref`, `set-aref`
- `setf`
- quasiquoting
- `gensym`