

Hy: Lista Interminable y Supérflua de Paréntesis

Guillermo Vayá Pérez

guivaya@gmail.com

Hy!!

Bienvenidos

Sobre mi

Guillermo Vayá Pérez

- E-mail: guivaya@gmail.com
- Twitter: [@Willyfrog_](https://twitter.com/Willyfrog_)
- Web:
- Irc: Willyfrog en freenode.net, Github y otros

Sobre Hy

- Lisp 1
- Embebido en python
- Soporta:
 - 2.6
 - 2.7
 - 3.2
 - 3.3



Getting Hy

```
pip install hy
```

- Ocasionalmente es mejor:

```
pip install -e <github repo>
```

¿Por qué lo Hyzo?

Autor

Paul Tagliamonte



@paultag

Origen

- Boston Python Group
- Inicialmente: Uso de AST de Python
 - Excasa documentación del AST
- Clojure

En general

- Los interpretes de Lisp son muy comunes
- Primer programa escrito en GO: ¡Un interprete de Lisp!

Objetivo

- Interoperabilidad Python-Lisp
- Aprender
- ¡Divertirse!

Hyntroducción

Introducción a Lisp en general
(Y a Hy en particular)

Homoiconicidad

listas

```
(primero resto)  
(primero (primero resto))
```

Código vs Información

- código

```
(+ 1 2)
```

- Información

```
"suma" "x" "y"
```

Tipos básicos

- int (incluye long):

```
3 1 4 5 6
```

- float:

```
3.3
```

- strings (pero son unicode):

```
"Hy Python!"
```

Y variables

```
(setv who "people")  
(saluda-a who)
```

Colecciones (mira mamá: isin comas!)

- Listas

```
[1 2 3]
```

- Diccionarios

```
{"Hy" "0.9.11" "Python" "3.3" }
```

- Tuplas

```
(, 1 2 3)
```

Funciones

Para invocarlas simplemente recolocamos los parentesis:

```
(len [1 2 3])
```

Para crear nuevas funciones:

```
(defun saluda-a [x]  
  (+ "hy " x "!"))  
  
(saluda-a "people")
```

Lambdas

```
(fn [x]  
  (+ x 1))
```


Clases e Instancias

```
(defclass Persona [  
  [--init-- (fn [name]  
              (setv self.name name)  
              None) ]  
  [name None]  
  [saluda-a (fn [quien] (print (% "%s saluda a %s" (, self.name quien)))) ]])  
  
(setv yo (Persona "Guillermo"))
```

Expresiones avanzadas de python

- list comprehension

```
(list-comp
  (, x y)
  (x (range 9)
   y "ABCDEFGH"))

; [(0, 'A'), (0, 'B'), (0, 'C'), (0, 'D'), (0, 'E'), (0, 'F'), (0, 'G'), (0, 'H'),
;  (1, 'A'), (1, 'B'), (1, 'C'), (1, 'D'), (1, 'E'), (1, 'F'), (1, 'G'), (1, 'H'),
;  (2, 'A'), (2, 'B'), (2, 'C'), (2, 'D'), (2, 'E'), (2, 'F'), (2, 'G'), (2, 'H'),
;  (3, 'A'), (3, 'B'), (3, 'C'), (3, 'D'), (3, 'E'), (3, 'F'), (3, 'G'), (3, 'H'),
;  (4, 'A'), (4, 'B'), (4, 'C'), (4, 'D'), (4, 'E'), (4, 'F'), (4, 'G'), (4, 'H'),
;  (5, 'A'), (5, 'B'), (5, 'C'), (5, 'D'), (5, 'E'), (5, 'F'), (5, 'G'), (5, 'H'),
;  (6, 'A'), (6, 'B'), (6, 'C'), (6, 'D'), (6, 'E'), (6, 'F'), (6, 'G'), (6, 'H'),
;  (7, 'A'), (7, 'B'), (7, 'C'), (7, 'D'), (7, 'E'), (7, 'F'), (7, 'G'), (7, 'H'),
;  (8, 'A'), (8, 'B'), (8, 'C'), (8, 'D'), (8, 'E'), (8, 'F'), (8, 'G'), (8, 'H')]
```

- uso de decoradores

```
=> (defn inc-decorator [func]
...  (fn [value-1 value-2] (func (+ value-1 1) (+ value-2 1))))
=> (with-decorator inc-decorator (defn addition [a b] (+ a b)))
=> (addition 1 1)
4
```

- generadores

```
(defn random-numbers [low high]
...  (while True (yield (.randint random low high))))
=> (list-comp x [x (take 15 (random-numbers 1 50))]))
[7, 41, 6, 22, 32, 17, 5, 38, 18, 38, 17, 14, 23, 23, 19]
```

Macros

- defmacro
 - quote (') quasiquote (`) unquote (~) y unquote splice (~@)
- Python en el compilador ()
- Tiempo de compilación vs. Tiempo de ejecución.

Imports

Clojure en versión Python:

- ¡Importar las baterías de Python desde Lisp!

```
(import json)
```

```
(import [datetime [datetime]])
```

Diferencias con lisp

- No hay reader macros
- Uso de librerías de Python
- Hereda de varios dialectos
- No hay
 - Uso de listas de python []

Si tiene cons

- kwapply
- Representacion en Python

Probando la Hydea

Veamos un pequeño ejemplo

hyndagando

Composición

- Python
 - AST
- Hy

Composicion (II)

- Hy
 - macros
 - utilidad
- rply (modificado)
 - Parser
 - Lexer
- Python (AST + lenguaje)
 - Compiler
 - Python VM

Compilación

Python -> Ast -> Bytecode

^ Hy

AST

- Es un arbol que contiene la estructura de los programas Python.
- Se puede usar para extender Python.
- Contiene meta informacion (posicion en el código)

Cosas a tener en cuenta

- No es una api estable => Puede cambiar entre versiones de Python.
- Subset de instrucciones.
- Hay que dar siempre una posicion en el código.
 - `ast.fix_missing_locations` (no sirve en Hy)
- Los errores son un poco más complejos.

Ejemplo AST

```
>>> import ast
>>> ast.dump(ast.parse("len([])"))
Module(body=[Expr(
  value=Call(func=Name(id='len',
                       ctx=Load()),
             args=[List(elts=[], ctx=Load())],
             keywords=[],
             starargs=None,
             kwargs=None)])])
```

Imports

- Pep 302 define hooks para los imports
- Originalmente pensado para poder importar ficheros comprimidos (zip)

Definiendo los tipos

Hy tiene sus propios tipos, que posteriormente se traducen a python

```
=> (defmacro show-your-type [x] (print (type x)) `~x)
=> (show-your-type {})
<class 'hy.models.dict.HyDict'>
{}
=> (show-your-type 1)
<class 'hy.models.integer.HyInteger'>
1
```

Algunos no son lo que parecen: HyDict

Macros

- No estan en el espacio normal de las funciones => No macro for python.
- Compilan funciones.
- Utiliza los tipos de hy
- Para importarlas se usa require.

```
(require hy.contrib.meth)
```

¿Quieres aHyudar?

Recursos

- Web/Docs:
- Github: (ver otros proyectos de hylang)
 - hy-mode para Emacs
 - Vim plugin
 - Flask
 - AppEngine
- Irc: #hy en freenode.org
- Pruebalo:

Herramientas

- `ast.dump(ast.parse(x))`
- `hy2py`
- `hy -spy`

Gracias

THE
SIMPLE ANSWERS
TO THE QUESTIONS THAT GET ASKED
ABOUT EVERY NEW TECHNOLOGY:

WILL <input type="checkbox"/> Hy MAKE US ALL GENIUSES?	NO ^{Of course}
WILL <input type="checkbox"/> Hy MAKE US ALL MORONS?	NO
WILL <input type="checkbox"/> Hy DESTROY WHOLE INDUSTRIES?	YES
WILL <input type="checkbox"/> Hy MAKE US MORE EMPATHETIC?	NO
WILL <input type="checkbox"/> Hy MAKE US LESS CARING?	NO
WILL TEENS USE <input type="checkbox"/> Hy FOR SEX?	YES
WERE THEY GOING TO HAVE SEX ANYWAY?	YES
WILL <input type="checkbox"/> Hy DESTROY MUSIC?	NO
WILL <input type="checkbox"/> Hy DESTROY ART?	NO
BUT CAN'T WE GO BACK TO A TIME WHEN—	NO
WILL <input type="checkbox"/> Hy BRING ABOUT WORLD PEACE?	NO
WILL <input type="checkbox"/> Hy CAUSE WIDESPREAD ALIENATION BY CREATING A WORLD OF EMPTY EXPERIENCES?	WE WERE ALREADY ALIENATED

Fuente:

Contacto:

- E-mail: guivaya@gmail.com
- Twitter: [@Willyfrog_](https://twitter.com/Willyfrog_)
- Web: