(= HY (+ PYTHON LISP))

Eléonore Mayola, PhD - @EleonoreMayola
Junior data scientist @MastodonC, organiser @PyLadiesLondon

About Lisp

- 1958: second oldest high-level programming language still in use today
- Created as a practical mathematical notation for computer programmes
- Became the most widely used language in Artificial Intelligence research
- Characterised by parenthesised lists:
  `(function argument1 argument2)`
- Today Lisp dialects are used for general purpose; Common Lisp, Scheme, Clojure, and Hy

Intro to Hy

Hy is a Lisp dialect that converts its structure to Python. Within Hy you have access to all of Python data structures and the standard library.
It's as easy as `(print "Hello, world!")`.
Try Hy in your browser at try-hy.appspot.com/

Calculations go from
```
(3.5 + 5.1 + 4.6) / 3
```
to
```
(/ (+ 3.5 5.1 4.6) 3)
```

If/else statement example:
```
=> (if (= 3 (+ 1 2))
... (print "This is true")
... (print "This is false"))
This is true
```

For loop example:
```
=> (for [i (range 6)]
... (print (+ "i equals " (str i))))
i equals 0
i equals 1
i equals 2
i equals 3
i equals 4
i equals 5
```

* See more code examples at **bit.ly/try-hy**
* Get started with Hy by following instructions at **docs.hylang.org/en/latest/quickstart.html**

A Hy programme

My example Hy programme analyses text files. You can find it at bit.ly/text-analysis-hy.
My imports : `(import os re pprint [collections [Counter]])`
The function performing the analysis is:

```
(defn analyse-texts [dirpath]
  (setv text-files (list-text-files dirpath))
  (list (map (fn [f]
               (setv filename f)
               (-> f
                   read-text
                   clean-text
                   remove-stopwords
                   (summarise-text filename)))
             text-files)))
```

"**defn**" the function definition, inspired from Clojure

"**setv**" sets a variable by binding a symbol to a value, a function...

"**fn**" the anonymous function

"**map**" returns an iterable that applies the anonymous function to each file in the text-files list.

"**->**" the thread first, inspired from Clojure. Enables function chaining w/o several levels of nesting.

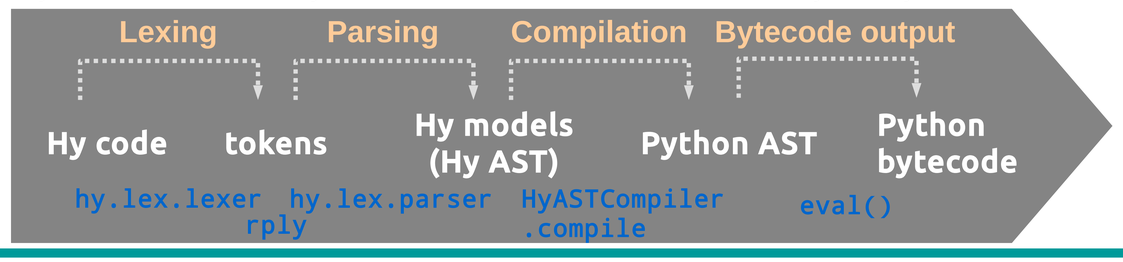
You can define a "main" function to reproduce the "if __name__ == '__main__'" behaviour.
By using `(defmain [&rest args] …)`, you can run a main from the command line with its arguments.

Hy under the hood

Basic steps of compilation:
Lexing → lexical analysis: breaks up the code into tokens
Parsing → syntax analysis: convert a sequence of tokens into a parse tree
Code generation → translate the parse tree into bytecode

AST (Abstract Syntax Tree): data structure used by compilers to represent the structure of the source code. It's the result of the parsing (syntax analysis) step.

Hy first translates to a Python AST which is then built down into Python bytecode.



Hy Models: a layer on top of Python objects representing Hy source code as data.
They define Hy objects that can add info to help the manipulation of the Hy source code.

Macros

References:

My repository for this poster:
github.com/Eleonore9/hy-python-lisp

- Hy's docs: docs.hylang.org/en/latest/
- More docs: github.com/hylang/hy/blob/master/docs/language/api.rst
- Hy's source code: github.com/hylang/hy
- Podcast.__init__ episode 23: pythonpodcast.com/hylang-developers.html
- Videos and blogposts: gist.github.com/Foxboron/4b87b5b85d6c5fc5db6c

Tools:
- Emacs Hy-mode: github.com/hylang/hy-mode
- Vim-hy: github.com/hylang/vim-hy
- Code analyser: github.com/hylang/hydiomatic
- Hy debugger: github.com/hylang/hdb