hylang

A short tour of a lisp embedded within python

Arch Lisp Meetup : Thursday, July 13, 2017

Features & Observations

- transforms Lisp code into the Python Abstract Syntax Tree
- interopt with python in both directions
  - call Lisp/Hy from Python code
  - call Python from Hy code
- Lisp syntax inspired by Clojure
- Can be a fun & fast way to start exploring Lisp
- Both 2.7.x and Python 3.x compatible (More portable than normal python!)

https://github.com/hylang/hy

Installation/Experimentation

(assuming you've already got python installed on your system)

```
# create a "playground"
mkdir hy-play && cd hy-play

# create virtualenv
virtualenv hy-venv

# activate the virtualenv
source hy-venv/bin/activate

# install it
pip install git+https://github.com/hylang/hy.git

# start up a REPL
hy
```

Quickstart

```
$ hy
hy 0.13.0+26.g5610d7d using CPython(default) 3.6.1 on Darwin
=> (print "Hy!")
Hy!
=> (defn salutations [name] (print (+ "Hy " name "!")))
=> (salutations "Dude")
Hy Dude!
=> <Ctrl+D> ; exit
now exiting HyREPL...
```

Can make executable lisp/hy scripts as well!

```
$ ./examples/hi.hy
Howdy!
```

Syntax

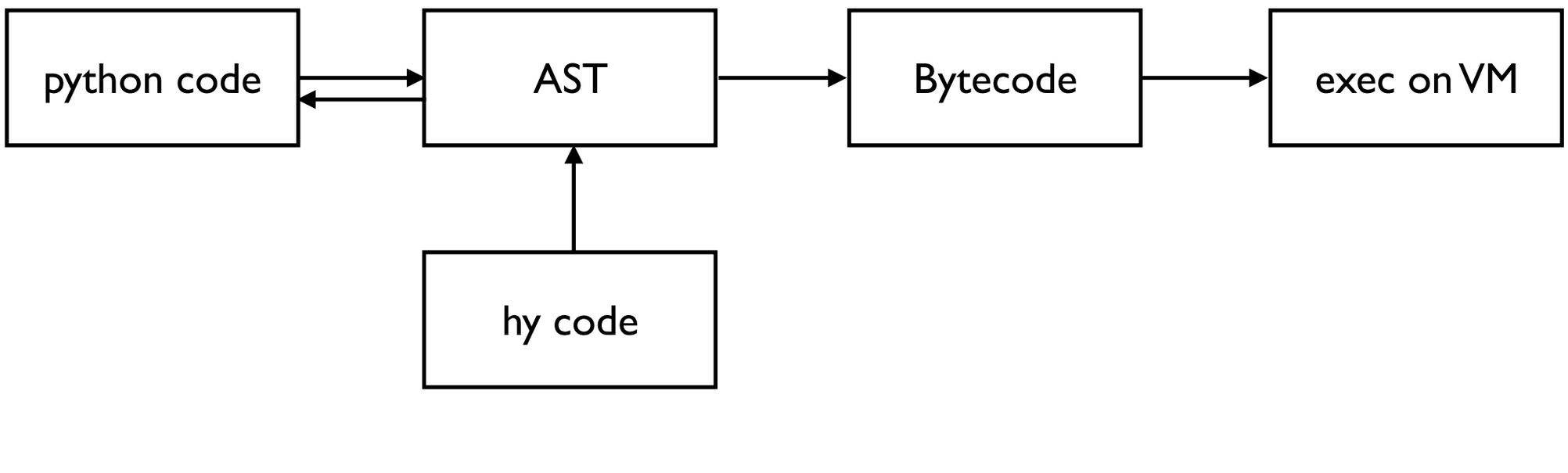
```
=> (.rstrip "foooo     ")
‘foooo'

=> (setv this-string "foooo     ")
=> (this-string.strip)
‘foooo'

=> (.strip this-string)
'foooo'
```

How does this work?

The Abstract Syntax Tree (AST)




```
 hy2py examples/hi.hy     # translate to python2
(hy2py3 examples/hi.hy    # translate to python3)
 hy2py -a examples/hi.hy  # dump AST logic
```

Data Types

```
; python lists
=> [1 2 3]
[1, 2, 3]

; python dictionaries
=> {"dog" "bark"
... "cat" "meow"}
{'dog': 'bark', 'cat': ‘meow'}
```

```
; keyword syntax
=> {:dog "bark"
... :cat "meow"}
{'\ufdd0:dog': 'bark',
 ‘\ufdd0:cat': 'meow'}
```

```
; python tuples
=> (, 1 2 3)
(1, 2, 3)

; python sets
=> #{3 1 2}
{1, 2, 3}

; extra: Fraction Literal – just like Clojure
=> 1/2
Fraction(1, 2)
```

Conditionals

Simple

```
(if (= 1 1)
  (print "if it's true")
  (print "if it's false"))
; "if it's true
```

Complex

```
(setv somevar 33)
(cond
 [(> somevar 50)
  (print "That variable is too big!")]
 [(< somevar 10)
  (print "That variable is too small!")]
 [True
  (print "That variable is jusssssst right!")])
; That variable is jusssssst right!
```

Looping

Basic

```
(for [i (range 3)]
  (print (+ "'i' is now at " (str i))))
; 'i' is now at 0
; 'i' is now at 1
; 'i' is now at 2
```

List Comprehensions

```
(setv odds-squared
(list-comp
  (pow num 2)
  (num (range 100))
  (= (% num 2) 1)))
```

(hylang)

```
odds_squared = [
pow(num, 2)
for num in range(100)
if num % 2 == 1]
```

(python)

Argument Passing

(python)

```python
def optional_arg(pos1, pos2, keyword1=None, keyword2=42):
    return [pos1, pos2, keyword1, keyword2]
```

(hylang: standard)

```
(defn optional-arg [pos1 pos2 &optional keyword1 [keyword2 42]]
  [pos1 pos2 keyword1 keyword2])
```

(hylang: keyword-style)

```
(optional-arg :keyword1 1
              :pos2 2
              :pos1 3
              :keyword2 4)
```

(hylang: dictionary-style)

```
(defn another-style [&key {"key1" "val1" "key2" "val2"}]
  [key1 key2])
```

(hylang: destructuring)

```
(defn multiarr [[x y] z]
  (+ x y z)

=> (multarr [1 2] 3)
6
```

*arg and **kwarg support

```python
def some_func(foo, bar, *args, **kwargs):
  import pprint
  pprint.pprint((foo, bar, args, kwargs))
```

(hylang) ⟶

```
(defn some-func [foo bar &rest args &kwargs kwargs]
  (import pprint)
  (pprint.pprint (, foo bar args kwargs)))
```

⟵ *(python)*

context managers / file handling

```python
with open("/tmp/data.in") as f:
    print f.read()
```

(python)

```
(with [f (open "/tmp/data.in")]
  (print (.read f)))
```

(hylang : using direct python calls)

```
(with [f (open "thing.hy")]
   (try
     (while True
            (setv exp (read f))
            (print "OHY" exp)
            (eval exp))
     (except [e EOFError]
            (print "EOF!"))))
```

(hylang : using the `read` function)

Classes

```python
class FooBar(object):
    """
    Yet Another Example Class
    """
    def __init__(self, x):
        self.x = x

    def get_x(self):
        """
        Return our copy of x
        """
        return self.x

# Usage
bar = FooBar(1)
print bar.get_x()
```

python

```hylang
(defclass FooBar [object]
  "Yet Another Example Class"

  (defn --init-- [self x]
    (setv self.x x))

  (defn get-x [self]
    "Return our copy of x"
    self.x))

;; Usage
(setv bar (FooBar 1))
(print (bar.get-x))

;; or
(print (.get-x (FooBar 1)))
```

hylang

Macros

```
(defmacro hello [person]
  `(print "Hello there," ~person "!"))
(hello “Human")

=> (hello “Human)
Hello there, Human !
```

```
(defmacro rev [code]
  (setv op (last code) params (list (butlast code)))
  `(~op ~@params))

(rev (1 2 3 +))

=> (rev (1 2 3 +))
6
```

hy <=> python interop

python calling hy code

file: greetings.hy

```
(defn greet [name] (print "hello from hy," name))
```

file: greet.py

```
#!/usr/bin/env python
import hy
import greetings

greetings.greet("Foo")
```

---

shell command:

```
python greet.py
```

hy <=> python interop

hy calling python code

```
(import os)

(if (os.path.isdir "/tmp/somedir")
  (os.mkdir "/tmp/somedir/anotherdir")
  (print "Hey, that path isn't there!"))
```

Multiple Packages Imports, choosing selective functions, & alt. namespacing

```
(import [functools [reduce]]
        [pprint [pprint]]
        [cytoolz [itertoolz]]
        [numpy :as np]
        [matplotlib :as mpl]
        [matplotlib.pyplot :as plt]
        [seaborn :as sns])
```

Need to use the `require` function to import macros from other modules

```
(require hy.contrib.loop)
```

useful functions & macros

Some Example functions:

```
., ->, ->>, apply, assoc, cons, cond, continue, fn, do, doto,
eval, first, last, rest, cut, for, get, nth, empty?, inc, dec,
list-comp, dict-comp, set-comp, quote, when, with,
with-decorator
```

http://docs.hylang.org/en/stable/language/core.html#

http://docs.hylang.org/en/stable/language/api.html#built-ins

http://docs.hylang.org/en/stable/extra/index.html

toolz / cytoolz

`toolz` : a pure python library that provides a suite of utility functions for data processing commonly found in functional languages

`cytoolz` : `toolz` ported to compiled C code via Cython (better performance on large data sets)

`pip install tools`
`pip install cytoolz`

Some Example functions:

```
frequencies, groupby, interleave, interpose,
mapcat, nth, countby, partitionby, assoc, merge,
```

http://toolz.readthedocs.io/en/latest/api.html

“practical” examples

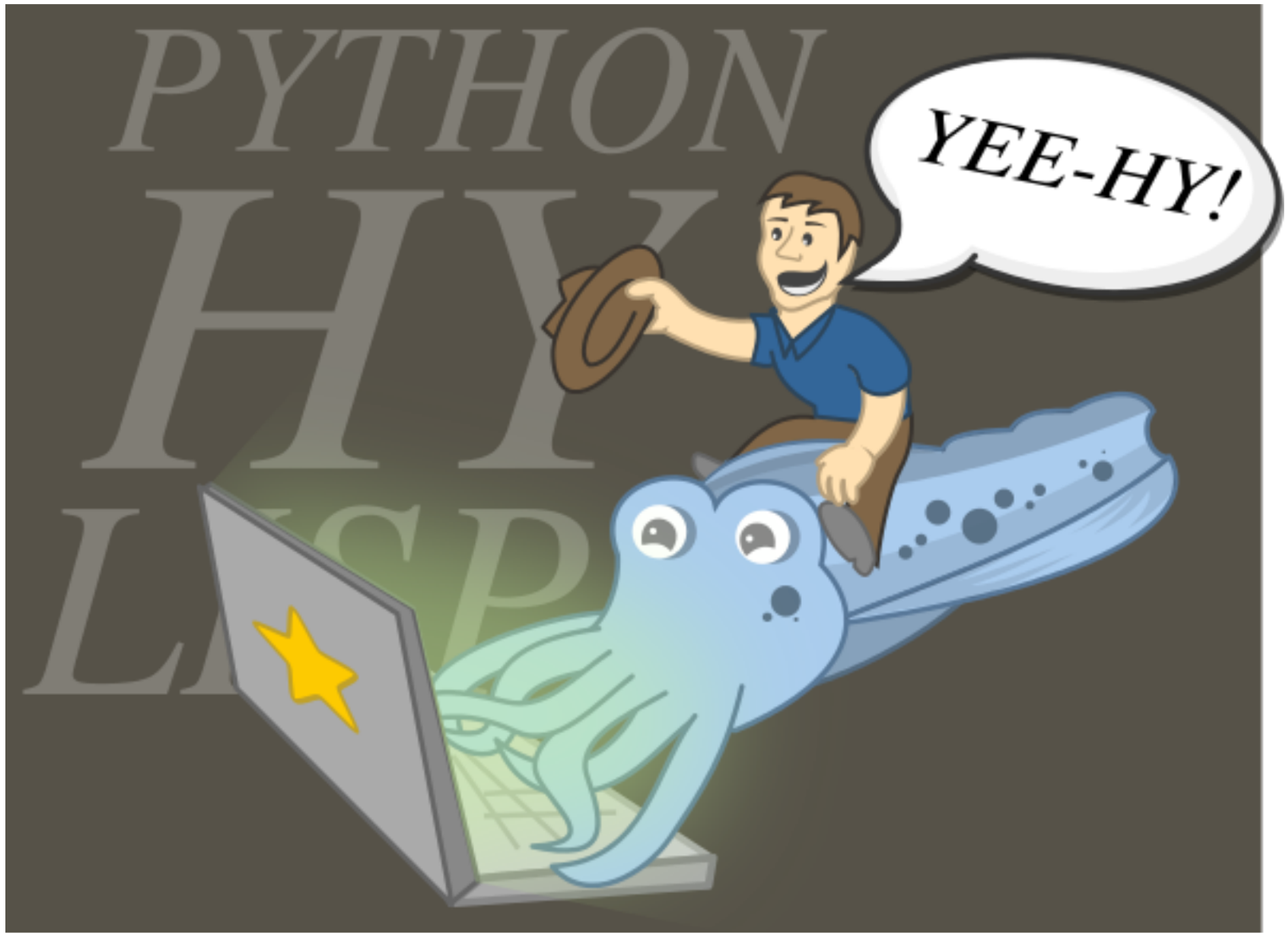
Further Reading & Watching

- Hylang's Documentation
  http://docs.hylang.org

- Hy Playlist on YouTube
  http://goo.gl/imYuG1

- OMG A Lisp that runs python
  https://goo.gl/e6eFqB

- Scientific Computing with Hy: Linear Regressions
  https://goo.gl/SfaMsd



Happy Lisping!