Lisp & Python: Hy

Agenda

1. My story
2. Python & Lisp: Peter Norvig's perspective
3. Python as Lisp
4. Making Python look like Lisp (but it is still really Python)
5. Hy:
    - Installation
    - Demo
    - Thoughts/Questions?
6. Pixie

My Story: from assembly to Clojure

- Z80 and Vax Assembler: real-time driver work
- Accidentally got a working copy of LOGO
    - Marketed as a language for kids
    - A real Lisp: it even has macros!
- LOGO → Lisp → Common Lisp
    - Cambridge Lisp for Atari 5200
    - XLisp Stat for work at Nielsen (1999)
    - SBCL after bumping into Peter Seibel's book
- Became interested in Python in the mid 90's (Python 1.4)
    - Started using it in earnest around 2000 because of pushback from colleagues
    - Very handy for analytics work (easy to parse CSVs, Numpy and data analysis toolkit
    - Became disenchanted as Python became bloated and started to add warts for meta- programming
- Clojure rekindled interest in Lisp
    - People are doing interesting things in it (Haskell, Scala folks do this as well)

Computers SHOULD be fun....

"I think that it is extraordinarily important that we in computer science keep the fun in computing. When it started out, it was an awful lot of fun. Of course, the paying customers got shafted every now and then, and after a while we began to take their complaints seriously. We began to feel as if we really were responsible for the successful, error-free perfect use of these machines. I don't think we are. I think we're responsible for stretching them, setting them off in new directions and keeping fun in the house. I hope the field of computer science never loses its sense of fun. Above all, I hope we don't become missionaries. Don't feel as if you're a Bible salesman. The world has too many of those already."

Alan Perlis (from the dedication page of *SICP* by Abelson and Sussman)

Peter Norvig: Is Python Lisp in Disguise?

Python can be seen as a dialect of Lisp with "traditional" syntax (what Lisp people call "infix" or "m-lisp" syntax). One message on comp.lang.python said "I never understood why LISP was a good idea until I started playing with Python." Python supports all of Lisp's essential features except macros, and you don't miss macros all that much because it does have eval, and operator overloading, and regular expression parsing, so some--but not all--of the use cases for macros are covered.

I came to Python not because I thought it was a better/acceptable/pragmatic Lisp, but because it was better pseudocode.... I think Lisp still has an edge for larger projects and for applications where the speed of the compiled code is important. But Python has the edge (with a large number of students) when the main goal is communication, not programming per se.

Python as a Lisp

Some key features of Python that make Lispers comfortable (didn't say happy...) in Python:

- Lists (and tuples) are a fundamental data structure
- Functions are first-class objects: in Python functions can:
    - Be created on demand
    - Be stored in a data structure
    - Be passed as an argument to a function
    - Be returned as the value of a function
- Python does *not* require the use of Object-oriented programming techniques
- Python is expressive: one can do a lot in a few lines
- Python has a REPL (and supports incremental programming beautifully)

Some things that get in the way (from a Lisper's perspective):

- No macros--really handy when creating DSLs
- Methods modify in place--disconcerting
- Very limited anonymous functions (lambda)

Aesthetic issue: Python doesn't look or feel like a Lisp...

Alchemy: Making Python look like Lisp

Python uses AST (Abstract Syntax Trees) internally

```
In [15]: import ast

In [16]: ast.dump(ast.parse("x = y + 2"))
Out[16]: "Module(body=[Assign(targets=[Name(id='x', ctx=Store())],
value=BinOp(left=Name(id='y', ctx=Load()), op=Add(), right=Num(n=2)))])"
```

We can compile and execute ASTs:

```
In [17]: eval(compile(ast.parse("x=42"),'<input>','exec'))

In [18]: x
Out[18]: 42
```

What if we transform Lispy code into Python ASTs? We get Python with a Lisp flavor--but this is still Python.

We get Hy*: a compiler that parses Lisp-like strings and generates Python ASTs.*

Getting started with Hy (by the book--this is more cumbersome than needed)

Hy documentation: http://docs.hylang.org/en/latest/index.html

1. Create a Virtual Env: (http://askubuntu.com/questions/244641/how-to-set-up-and-use-a-virtual-python-environment-in-ubuntu)
2. Activate the Virtual Environment:
3. Install Hy: `pip install hy`
4. Start a REPL with `hy`:

```
afmoreno@mercury:~$ ls ./.virtualenvs/
get_env_details  postdeactivate    preactivate      prermvirtualenv
hy_python        postmkproject     predeactivate    test
initialize       postmkvirtualenv  premkproject
postactivate     postrmvirtualenv  premkvirtualenv
afmoreno@mercury:~$ workon hy_python
(hy_python)afmoreno@mercury:~$ hy
hy 0.10.0
=>
```

Start hacking!

Getting started with Hy (not by the book)

1. Install `conda` from Anaconda.com
2. Create a new conda environment: `conda create --name hy-env`. I would add additional packages such as `matplotlib`, `pandas`, `seaborn`, etc. Refer to the `conda` documentation
3. Activate environment: `source activate hy-env`
4. Install Hy Jupyter kernel:
    - `pip install git+https://github.com/Calysto/calysto_hy.git --user`
    - `python -m calysto_hy install --user`
5. Start Jupyter notebook: `jupyter notebook`
6. You can kill the Jupyter server in the terminal (C-c C-c) and then deactivate the environment with `source deactivate`

Additional Links

1. Jupyter Notebook: http://jupyter.org/
2. Anaconda Python distribution: https://www.continuum.io/downloads