AI Computer Vision

Machine learning, natural language processing and ShapeLogic

Tuesday, February 5, 2013

Scala vs. Haskell vs. Python

Functional programming is on the upswing, but should you bet your career on it, or is it a short-lived technology fad?

I have long wanted to use functional programming professionally and for the last year I have. Mainly Scala, written in Haskell style, plus some real Haskell programming.

Here is my impression of Scala and Haskell compared to my benchmark language, Python.

Scala

Scala is a functional object oriented hybrid language running on the JVM. It was created by Martin Odersky in 2003. Scala took Java / JVM and organized it nicely according to a few orthogonal principles.

Working in Scala has been a pleasure, there is a lot to like:

- You have easy access to the giant world of Java libraries
- Lot of libraries written for Scala
- Very fast only 2 to 3 time slower than C
- Big ecosystem
- Easy to define a DSL in Scala so you can do everything in Scala
- Very advanced type system
- Adapted in the industry by: Twitter, LinkedIn, Foursquare, ...
- Scala is the most adapted functional language
- Web frameworks: Play, Scalatra, any kind of Java Servlets
- Scalding: a very nice framework for Hadoop programming
- Akka: an Erlang style actor system
- Mixin composition
- Good GUI with Swing and JavaFX
- SBT the best build tool I have used
- Scala is a full stack multi purpose language

Issues

- It is very complex
- It is a kitchen sink language
- Confusing to keep Scala collections and parallel Java collections apart

Eclipse Plugin Scala IDE for Eclipse
The Scala Eclipse plugin is very solid, but not quite as good as the fantastic Java support.

- Syntax highlighting
- Code completion
- Debugger
- Shows compile errors with explanation
- Rudimentary refactoring
- Jump to definition

**Monad and Applicative Functor**

Two very important concepts in functional programming are monad and applicative functor.

The best reference I found was: [Learn You a Haskell for a Great Good!](https://learnyouahaskell.com/).

A monad gives you simple ways of composing different operations. First it seems like an odd principle. Understanding monad took me several months.

In UNIX and OS X you can create complex program by piping simple commands together. A monad generalizes this a lot.

Once you understand the monad you will see monads pop up so many places. The monad is an amazingly powerful construct.

The last place I found monads unexpectedly showed up was in asynchronous programming, e.g. used in AJAX. You send an external request and you do not block but you have a callback for when the result comes back. This is efficient but messy to program especially if you have a chain of requests to process and you have to have a lot of callbacks floating around. You can do this type of calculations using a future / promise, and luckily a future is a monad so you string a long list of operations after each other in a very simple way.

**Scalaz**

Scalaz is a Scala library that replicates a lot of Haskell constructs, at the cost of being similarly hard to understand.

You can work with monads in Scala without using Scalaz since the "for-statement" in Scala is syntactic sugar for monadic "for-comprehension".

I have programmed Java in a functional style both professionally and for my open source project. It is possible but it is rather verbose and clunky. Scala is much more powerful, simpler and cleaner than both Java approaches, and Scalaz is a big step up from Scala.

When I started programming in Scala I read a really funny blog post called [Truth about Scala](blog.samibadawi.com/2013/02/scala-vs-haskell-vs-python.html) that describes how a team starts to use Scala and first they are excited, but it quickly descends into a death spiral of complexity. I was concerned with this and tried to keep my code as simple as possible and avoid Scalaz for a long time. I would advise
other to become very comfortable with Scala before starting to work with Scalaz.

Haskell

Haskell is a strongly typed, lazy, pure functional programming language. It is an academic research language created by a committee in 1987. One reason that I got into Haskell was in order to understand monads and applicative functors, they are important constructs in Haskell and category theory.

There is a steep learning curve for Haskell. Maybe it is more like a hump you have to get over. Just getting to basic proficiency is hard. It took me around one year of low intensity studying, but one day it just made sense.

- Haskell now has a lot of libraries
- Libraries and dependencies are handled by Cabal
- It is fast only 2 - 3 times slower than C
- Great concurrency
- Repa native parallel numerical array processing
- Very small language
- Very pure
- Very terse code
- Very advanced type system
- Hoogle a Haskell search engine
- Great web frameworks Happstack, Snap and Yesod

Issues

- Bad GUI support
- Module system is crude

Hoogle, a Search Engine for Haskell

A colleague told me that when he needed a function he would write it out its signature and put it into Hoogle and often it will take him to the function that he needed. First time I tried it and it actually took me to a function that solved a bigger part of the problem than what I was looking for.

When I searched Hoogle for this function signature:

```
(a -> Bool) -> [a] -> [Int]
```

I got these results in EclipseFP:
The Haskell Eclipse plugin is quite good:

- Syntax highlighting
- Cabal integration
- Hoogle integration
- Code completion
- Debugger
- GHCI integration with automatic reload

Python

*Python* is a high-level language built on ideas from functional, imperative and object oriented programming. It was created by Guido van Rossum in 1989.

For many years Python was my favorite language. It is a language for kids and also for scientists and a lot of people in between.

- Python is probably the easiest language to learn
- It took me a day to learn well enough to use
- Very minimal language
- Very terse code
- Excellent wrapper language
- Many implementations: CPython, Jython (JVM), IronPython (CLR), PyPy
- Good bindings to numerical packages: NumPy, SciPy
- Used in computer vision since *OpenCV* choosing Python to be its scripting language
- Used in natural language processing due to the *NLTK*
- Great web frameworks: Django, TurboGear, CherryPy

**Issues**

Python is not quite a full stack language there are a few missing pieces:

- Bad GUI support
- Low-level numerical programming had to be done in external packages
- Concurrency
- Speed around 50 times slower compared to C

**Eclipse Plugin PyDev**
I like PyDev it has:

- Syntax highlighting
- Code completion
- Debugger

Best Programming Language for Kids

If a kid can understand a technology it is well designed. My daughter is turning 5 and I am thinking about what language I should introduce her to first.

Python

My first inclination was to teach her Python since it is the simplest, but it needs to give immediate visual feedback. Python's lack of a good GUI is a problem.

Haskell

I have also been tempted to show her some Haskell to teach her good habits in a pure and minimal language. But if I tell her that:

"A monad is just a monoid in the category of endofunctors"

she will walk away or scream.

Scala

Kojo is a LOGO like graphical turtle programming environment written in Scala. Scala's type inference makes it simpler for kids who will not have a good concept of types.

My daughter plays with Kojo and she likes it. She comes and asks me if we can do the turtle?

So unexpectedly, Scala the biggest language, was the most kid friendly language. Based on a very small sample size.
Haskell is using plenty of concepts from category theory. E.g. the monad. In my quest to understand it I started to study category theory.

Category theory has been called: "Abstract nonsense", both by its practitioners and critics. And for very good reasons. It can suck you into a black hole of abstraction.

**Category Theory Introductions**
You do not need to understand category theory to program in Haskell or Scalaz, but if it helps you here are a few introduction videos.

Dominic Verity presents a gentle introduction to Category Theory:

http://vimeo.com/17207564

Dominic Verity on Category Theory (Part 2)

Error792's category theory class, currently there are 5 parts

---

**Math and Programming**

I have often said that there is no connection between math and programming. The only math you need to program is counting, and occasionally, addition. I felt:

*Programmers are the grease monkeys of today*

*We move some data around and throw it on webpages*

After working in Scala and Haskell I have changed my tune:

*When you program in Scala you feel like an engineer*

*When you program in Haskell you feel like a mathematician*

---

**Adapting Haskell and Scalaz for a Team**
Using Haskell and Scalaz takes a special mindset and a lot of dedication. I have been very lucky to work at a place that has attracted physicists, mathematicians and theoretical CS people.

If a big part of your team does not have these qualities you risk wasting time and chasing developers away.

On the other hand if your team is using Haskell or Scalaz you will attract this brand of developers.

Conclusion

I had high expectation when I started using functional programming full time, but I have been disappointed by new technology many times before. Functional programming met my high expectations. It has been challenging and very enjoyable.

I was a C++ programmer for 8 years, and considered C++ the one true way for high speed, high level programming.
Recently I looked at a code sample written in C++ and it hurts my eyes: Filled with boilerplate and state.

Functional programming is addictive and will make you spoiled

Functional programming is here to stay. It has been an important part of C# since v3.0. It is finally getting added to Java in Java 8 coming out soon. The classic functional languages LISP or ML are are the basis of: Clojure and F# that have thriving community and are used in industry. The time has come to invest some time in understanding functional programming.

Python

I enjoy Scala and Haskell more than Python, but Python seem to be the language that I always go back to. It is a power tool that adds very little weight to your programmer's toolbox. You get high return on investment with Python, while with Scala and especially Haskell you have to invest a lot and for a long time before you break even.

Scala

Scala is now popular enough that you can get a job doing it. Moving from Java or C# to Scala is pretty easy. Since you can start programming Scala like Java. Scala is a big and complex language with a big ecosystem and it takes months to get a deeper understanding. Scala is substantially more powerful than Java 7, but Java 8 has supposedly taken a lot of ideas from Scala.

Haskell

Haskell is definitely the road less traveled, but it is a road, not a trail. It is an academic research language created in 1987. Recently it has started to break into the mainstream. There are a few jobs in Haskell. Gaining basic proficiency in Haskell is quite hard, but afterwards other languages look a little clunky. Writing Haskell feels like doing math.

Scala vs. Haskell

Scala is a safer bet for most programmers, since it is better adapted to more tasks, and you can approximate Haskell pretty well with Scalaz. Scala has a very advanced type system to handle its object oriented features.

Haskell appeals to functional language purists, mathematicians and category theorists. Esthetically I prefer Haskell. It is terser and the type inference is better.

In most cases external factors would dictate whether Scala or Haskell would be a better fit for your project.

Haskell vs. Python

Haskell and Python have a lot in common:

- Minimalistic languages
- White space delimited
- Very terse

Haskell and Python have a lot in common:

- Minimalistic languages
- White space delimited
- Very terse

Haskell and Python have a lot in common:

- Minimalistic languages
- White space delimited
- Very terse
AI Computer Vision: Scala vs. Haskell vs. Python

- List comprehension
- Important tuple type
- GUI binding to wxWidgets, GTK

Haskell is statically typed and optimized towards purity and speed.

Python is dynamically typed and optimized towards pragmatism and simplicity.

Posted by Sami Badawi at 8:13 AM

Labels: category theory, education, functional programming, haskell, java, monad, Python, scala, scalaz

10 comments:

Maverick said...
Regarding python, most of the underlying numerics code is written in C (so it runs at full speed and you can drop the gil via "no-gil"), so python simply acts as a dynamic interface that sits on top of C functions. Most of my code now runs in the ipython notebook, which is a web-based GUI that runs in the browser. So my code runs on amazon, but I can edit/run remotely via the notebook. click here for a static (non-running) example: https://bitly.com/103OS2k

February 5, 2013 at 9:11 AM

Alvaro said...
Awesome post. It completely matches up with my experience with Scala and Haskell (I have barely used Haskell though) and my kids experience (7 year old twins) who also play with Kojo and who I have tried teaching a very basic bit of lisp and haskell.

February 12, 2013 at 10:22 AM

Sami Badawi said...
Hi Alvero,

Thanks I am happy to hear that I am not the only crazy dad that try to teach kids functional languages.

February 12, 2013 at 10:41 AM

Isaac Gouy said...
http://scratch.mit.edu/

April 8, 2013 at 10:53 AM

Sami Badawi said...
Thanks Isaac,

I have looked at Scratch, and it looks good. It is very GUI oriented.

April 8, 2013 at 1:05 PM

Avner Levy said...
Thanks Sami.
The choice of a language is dictated by what type of task you have to perform. I am using Python, and seen the large number of libraries I can use especially in ML (nltk.org), I am not ready to migrate my work to another language anytime soon.

(Then why am I reading your post? :) I had the same intent with my daughters. It's nice if there can be something attractive to show them as a result of their programming, but not essential imo. They were (they're now in secondary school) very happy to 'build a calculator' using a computer made like a human: eyes (reading an input from the console), brains (computing, comparing and branching), and arm (writing the result to the console). I used Java then.

May 18, 2013 at 9:26 AM
Jimi said...
I'm not sure I know what you mean by bad GUI support. Python's PyQt has a very strong following as well as making it incredibly easy to write GUI applications. In addition the folks at Kitware provide substantial Python support for their VTK visualization toolbox.
August 9, 2013 at 9:41 PM

Sami Badawi said...
Hi Jimi,

I got excited when Nokia changed the the Qt 4.5 license to LGPL. Finally an open source cross platform GUI with some muscle behind it. I was less happy when Nokia abandoned Qt.

PyQt is nice but idiosyncratic and relatively heavyweight. I have not looked at it for a while.
August 9, 2013 at 10:32 PM

Thefourtheye diVi said...
Thanks a lot for this awesome post. I wanted to learn a functional language. I believe haskell is the way to go, now. But, I would like to consider Erlang as well. Suggestions?
August 31, 2013 at 12:30 PM

Shannon Behrens said...
Useful. Thanks!
October 1, 2013 at 10:32 PM

Post a Comment

Links to this post
Create a Link

Newer Post
Home
Older Post

Subscribe to: Post Comments (Atom)