# Lisp is sin

**Confluence**

Over the last few days, there has been one thought running in my head - 'All roads lead to Lisp'. I've tried to get it out of my head but like a tune from the radio, it has firmly lodged itself into my head and refuses to let go. Why this state? Call it a confluence of several items. In particular, these 3.

- Joel Spolsky goes off and writes one of his best articles ever called '[The Perils of Java Schools](http://www.joelonsoftware.com/articles/ThePerilsofJavaSchools.html)' which talks about the perils of not learning Lisp. As someone who has been playing catch-up with functional programming since I wasn't taught Lisp at college, I definitely agree. I forwarded that article to some senior folks internally - which in turn lead to some interesting conversations

- I've been working on a prototype of something at work - where I started off the usual imperative programming style. Realizing that this wasn't going to scale, I rewrote it in functional style making full use of C#'s anonymous delegates and iterators. When I had to make a major change, it was \*so much\* easier to add in extra functionality since I was just passing closures around.

- After 2 hectic work-filled weeks, I took advantage of a long weekend to curl up with a book which has been called the best book on programming ever - Peter Norvig's [Paradigms of Artificial Intelligence Programming - Case Studies in Common Lisp](http://www.amazon.com/gp/product/1558601910/002-6413815-3000828?v=glance&n=283155). I have read SICP but the 300 or so pages I've read of Norvig's book have left a greater impression on me than SICP. Norvig's book is definitely one of those 'stay awake all night thinking about it' books. For one, the Lisp code is much easier to read than the code in Paul Graham's books (which is partially due to my lack of Lisp knowledge, of course).

This post is an attempt to put all my thoughts about Lisp in one place so that I can refer people to it when they ask me my opinion on whether they should write something in Lisp or Scheme. I swear that when I thought of this title, I didn't mean it in connection with Don Box's article called '[Scheme is love'](http://msdn.microsoft.com/msdnmag/issues/05/10/EndBracket/default.aspx) :-). Though there's a disclaimer link, I want to call this out explicitly - these rants are mine alone and not Microsoft's opinion. So don't assume they are!

**Love and Hate**

I have a love/hate relationship with Lisp and Scheme. Over the last 2 years, I've tried to convince myself to move to Lisp and Scheme away from Python as my primary language for doing 'fun stuff'. However, every time I have done so, I've run into blockers - almost always the same ones the reddit.com folks [ran into](http://reddit.com/blog/2005/12/night-of-living-python.html) - the lack of libraries which work across Lisp implementations (though the Lisp folks are [working](http://www.findinglisp.com/blog/2005/12/best-of-finding-lisp-2005.html) on fixing some of the issues - issues like the one mentioned in the last para [here](http://www.goesping.org/archives/2005/11/27/thoughts-reading-practical-common-lisp/)).

The first time I used or rather tried using Lisp was when I wrote a crawler back in college. I wound up writing it in Python because of the sheer difference in 'impedance' to do web programming. By 'impedance', I mean the amount of disjointed knowledge that you need to wrap your head around before you can write your code. In Python, it was just a question of importing urllib2 and a couple of other libraries and I was good to go. I eventually wound up using .Net to make use of Lucene.Net.**1**

After struggling to resolve the million and 1 Lisp dialects out there, I went another path - the Scheme way. This came about due to the after effects of reading of [SICP](http://mitpress.mit.edu/sicp/full-text/book/book.html) (a few conversations with [Roshan](http://www.thinkingms.com/pensieve/) also had something to do with it). I installed Dr Scheme and had a lot of fun. I remember writing Scheme code in college class for fun when the lectures got boring (which they were almost all the time).

My dabblings with Scheme lead me to another path - the implementation of virtual machines which eventually lead to [Smoke](http://smokevm.sourceforge.net/) - but that's a story for another day. However, as a result, I moved away from Scheme and to the land of VirtualAlloc and C++.

After joining Microsoft, I have played around with Lisp and Scheme on and off - usually inspired after spotting Paul Graham's book on my shelf or after a conversation with [Vishnu](http://vishnuvyas.wordpress.com/) about his Lisp experiences.

At work, code is in C# or C++ - the day when Visual Studio is written in functional programming is far away **2** :-). So it was inevitable that I would experience functional cravings from time to time when writing code. I've written very little code that ships inside Whidbey but a lot of prototyping, etc - and most times, this craving would be satisfied by C# 2.0. Though the syntax wasn't as nice and clean, it would usually get the job done. Of course, most times I just wanted map/filter and closures - all of which you can do using anonymous delegates (which are close enough to closures for me). And in recent times, Linq has kept me happy.

I could never quite make up my mind about Lisp - I used to oscillate between "Wow - I love the way my brain hurts. This is so much fun" to "Normal people will find this too tough. Lisp weenies!".

**The one where Google's director of research influences a Microsoft employee**

After a long wait, Norvig's book wound up on my table a few days back. Due to [work](http://blogs.msdn.com/somasegar/archive/2005/11/08/490694.aspx), I couldn't get started on it till a couple of days ago - and since then, I've spent almost all of my free time reading it.

What grabbed me was not the Lisp usage in the book. The code could have been written in [Intercal](http://en.wikipedia.org/wiki/INTERCAL_programming_language) (well -maybe not) - but the beauty doesn't lie in the language - it lies in the style that the language promotes. The sheer beauty of it is dazzling.

 I remember reading the following piece somewhere about Lisp and it struck a chord back then in me. Its about the progression of a Lisp programmer - the newbie realizes that the difference between code and data is trivial. The expert realizes that all code is data. And the true master realizes that all data is code.

Nowhere is this more apparent than the way Norvig [tackles](http://www.norvig.com/paip/README.html) the [General Problem Solver](http://en.wikipedia.org/wiki/General_Problem_Solver). I thought for a second as to how I would have tackled this in C# (or Python). I would never have gone for the data driven approach that Norvig uses - I would have gone for a more imperative approach with a bigger distinction with the data that makes up the individual ops and the code that executes it.

In *The Practice of Programming*, Rob Pike says something akin to 'document your data structures rather than your algorithms'. In Norvig's work, I could see the code flow out of the data structures. It really is a new way of thinking (at least for me) - to let your data drive your program rather than the other way about. I use the word 'grok' sparingly but this is something you need to 'grok' to truly understand. But when you do, the 'plop' that goes off inside your brain makes it hurt a lot. And expands it at the same time too.

Why does Lisp encourage this kind of programming? S-expressions. Since your code is data in itself, you find it easier to jump between the two. Your code \*is\* data - so you can change your code at runtime and you are in fact, encouraged to do so. Now, the same thing is possible in other languages as well - self-modifying programs are possible in C++ (and with some hackery, in .NET as well without using Reflection or LCG). However, getting them working, debugging and maintaining them is a feat in itself. **3**

 Joel hits the nail on the head when he says that today's students are missing out by not having their heads hurt due to Lisp. He talks about Microsoft lagging behind Google and though he doesn't directly say it, one comment I've heard from folks is that Google are better suited to do Lisp-style programming due to having people like Peter Norvig. I'm not sure I agree - with folks like Patrick Dussud **4** architecting the CLR, we have sufficient Lisp influence 🙂

If you want to be a programmer and if you don't know Lisp, you're missing out on something. The sad part is - most people do not even realize this. And the fault isn't even theirs...

**Lisp and the common man**

Here's the part that'll get me flames. This is not a laundry list of things that are wrong with Lisp - I won't even pretend to be good enough something like that. This is a list of things that have blocked me and people I know from Lisp. Of course, the real reason may be just that we are not smart enough but my ego won't permit me to consider that an option 🙂

In Visual Studio, we look at 3 distinct categories of programmers. We call them [Mort, Elvis and Einstein](http://wesnerm.blogs.com/net_undocumented/2003/09/who_are_you_mor.html) - the 3 personas we take into consideration whenever we create any developer technology. What is difficult for some of us geeks to comprehend sometimes is - all 3 of them are equally important. When you look at technology like Windows Workflow Foundation, one of the driving forces was to let non-geeks build software. Not everyone needs to be a Raymond Chen or a Dave Cutler. Not everyone needs to understand the difference between the various GC algorithms. However, everyone needs the ability to be productive. And everyone needs to be able to get software working without needing a CS degree.

In a way, this is a rebuttal to Joel Spolsky's post on Java schools and an extension of [this post](http://dotnetjunkies.com/WebLog/sriram/archive/2004/11/18/32707.aspx) from my past life. We cannot afford to restrict software development only to those who know Windows kernel internals or those who can understand what a continuation is. It's not that other people are not smart - they just have better things to do. That's the key piece of understanding I find missing sometimes.**5**

I frequently see on Slashdot "Windows is designed for stupid users".  That is quite insulting to the millions of moms and dads, teachers and laywers and people from other walks of life who use Windows or even the Mac. If we mandated that every new user understand Windows' command line syntax or Emacs, we would have failed as an industry - we would have locked out the rest of the world.

Where am I going with all this? Lisp \*sometimes\* tends to ignore the common man - favoring giving the true hacker power rather than making it easy for the Morts of the world. I understand how it came about - when you write tools for yourself, you are the only user you have in mind - you and people like you. Joel talks about Java schools making it easy for students. However, look at the other side - if you don't make it easy for these students, they would never be exposed to the wonderful world of programming. I can attest that most of my college classmates wouldn't have touched CS with a pole if it had involved the stuff Joel talks about.

That's a hard problem to solve. What would you rather go for? An elite few armed with the knowledge to write great software? Or enabling the masses by making it simpler for them? A cop-out would be to say 'Find a middle way'. I'm not sure whether there is a middle way.

Python's battle cry of 'batteries included' is worthy of emulation by the makers of other programming systems **6**. .Net provides similar functionality but we don't have such a catchy tag-line :-). A close competitor would be the presence of standards (defacto, ECMA/ANSI/ISO blessed) for Python, Ruby and .Net. If you write a piece of code against a library, you can be sure it works across versions and variants. I know that if I write a C# program today that it can be called by a Boo program which in turn can be called by IronPython. In Lisp, it is often a challenge to find a) libraries that work across dialects b) libraries for common tasks which suit you. Scheme  solves the dialects problem - however, though it is incredibly fun to code in and despite the presence of stuff like the [SRFIs](http://srfi.schemers.org/), it doesn't seem to enjoy the popularity of CL. I'm not sure why that is though - maybe it would be a good thing if that changed. A language with such a short specification is a true work of art 🙂

The good news is that the Lisp community seems to be awoken to some of these issues - the reddit.com incident seems to have been the proverbial straw. By looking at stuff  like [this](http://www.findinglisp.com/blog/2005/06/ilc-2005-wednesday-report-late.html) which involves McCarthy himself, perhaps change is imminent.  I definitely look forward to the day when I can write defun within Visual Studio and have the full blown support of .net and the IDE. Slime+ XEmacs are just not for me 🙂

**The new Lisp**

Paul Graham says in 'On Lisp' *"The question is not whether Lisp..but rather 'when'"*. He also talks about Python and how Lisp features have made their way into Python. This brings forth an interesting question - will the next Lisp be Lisp itself? Or some other language? If you take a look at the programming language landscape, Python seems to be the frontrunner now in terms of bringing Lisp features to mainstream programming.

However, there are 2 little programming languages which may well prove to be the dark horses in this race. They are C# and VB and with folks like Anders Hejlsberg working on them, I wouldn't bet against them. Take a look at C# 2.0 anonymous delegates or Linq - they bring to millions of programmers what was once only available to Lispers (and in some cases, like Linq, stuff that Lispers wouldn't have had without some major effort).

After looking at Norvig's implementation of memoization in his book, I set out today to do the same in C# 2.0. Here's what I wound up with after 10 minutes.

public delegate TReturn Function<TParam, TReturn>(TParam param);

static Function<TParam, TReturn> Memoize<TParam, TReturn> (Function<TParam, TReturn> func)  
{

Dictionary<TParam,TReturn> memoDict = new Dictionary<TParam,TReturn>();  
return delegate(TParam arg)  
{

    TReturn result = default(TReturn);  
  
    if(memoDict.ContainsKey(arg))  
    {  
        result = memoDict[arg];  
    }  
    else  
    {  
        result = func(arg);  
        memoDict[arg] = result;  
    }  
    return result;

};

}

Is it as pretty as the Lisp counterpart? Probably not. But I'm willing to bet that a lot more developers will be able to understand this since this is in a programming language they understand well. And it gets the job done in most cases.

Maybe C# is the next Lisp. That would be a sight to behold 🙂

**Lisp - The Temptation**

I was on vacation a couple of weeks ago at my parents' house in Chennai. My dad and I share a love for James Bond movies so my dad had bought a set of DVDs containing all the Bond movies in existence. I can't help but strike a politically incorrect analogy - Lisp is like the [villainesses](http://en.wikipedia.org/wiki/Bond_girl) present in the Bond movies. It seduces you with its sheer beauty and its allure is irresistible. A fleeting encounter plays on your mind for a long,long time. However, it may not be the best choice if you're looking for a long term commitment. But in the short term, it sure is fun! In that way, Lisp is...sin.

***Notes:***

1. I took a look at Lupy and PyLucene and liked neither. However, I hear PyLucene has made great strides - Mitch Kapor's Chandler uses it if I remember correctly. So - any college students out there, don't trust anything I say.

2. A lot of people have asked me what language is used to write Visual Studio. The short answer is - COM filled C++ and C# form most of the code. However, our internal tools use a mix of batch scripting, Perl, etc. I was permanently mentally scarred the first time I saw an entire program written in batch file syntax - it makes Perl look elegant!

3, In AOCP, Don Knuth talks about a self modifying random number generator - which later turned out to be buggy!

4. Patrick Dussud worked on TI's Explorer project and with Richard Gabriel on Lucid. He caused a bit of a stir sometime back with his [talk](http://www.international-lisp-conference.org/2005/speakers.html#patrick_dussud) to the Lisp community entitled 'Re-inventing Lisp for Ubiquity'. Review [here](http://www.findinglisp.com/blog/2005/06/ilc-2005-wednesday-report-late.html). Another source of information on Patrick is from Richard Gabriel's book *'Patterns of Software'* which you can read at <http://www.dreamsongs.com/Books.html>. Even if you don't care about Patrick, do read the book - it is an awesome piece of work covering everything from the quality missing in software to Lucent, the story of XEmacs and the rise of Jamie Zawinski.

5. I find that in some of Paul Graham's writings - like at <http://www.paulgraham.com/avg.html>.  I love his work - but I sure would love to see him work in Apple or on the VB team . Something that people don't take into consideration is that the 'pointy haired boss' has to think of finding people to maintain your code - and when the company grows, finding more people to work on it. Paul's rebuttal to that is at <http://www.paulgraham.com/icad.html> - I would like to see that updated after reddit.

6. I'm not sure what the right word is for programming language + libraries+ tools+community so I'm using 'system' as a substitute.