[Stevey's Blog Rants](http://steve-yegge.blogspot.com/)

RANDOM WHINING AND STUFF.

**Friday, March 17, 2006**

Math For Programmers

I've been working for the past 15 months on repairing my rusty math skills, ever since I read a [biography](http://www.amazon.com/o/asin/0262011212) of [Johnny von Neumann](http://en.wikipedia.org/wiki/Von_Neumann). I've read a huge stack of math books, and I have an even bigger stack of unread math books. And it's starting to come together.   
  
Let me tell you about it.   
  
**Conventional Wisdom Doesn't Add Up**  
  
First: programmers don't think they need to know math. I hear that so often; I hardly know anyone who disagrees. Even programmers who were math majors tell me they don't really use math all that much! They say it's better to know about design patterns, object-oriented methodologies, software tools, interface design, stuff like that.   
  
And you know what? They're absolutely right. You can be a good, solid, professional programmer without knowing much math.   
  
But hey, you don't really need to know how to program, either. Let's face it: there are a lot of professional programmers out there who realize they're not very good at it, and they still find ways to contribute.   
  
If you're suddenly feeling out of your depth, and everyone appears to be running circles around you, what are your options? Well, you might discover you're good at project management, or people management, or UI design, or technical writing, or system administration, any number of other important things that "programmers" aren't necessarily any good at. You'll start filling those niches (because there's always more work to do), and as soon as you find something you're good at, you'll probably migrate towards doing it full-time.   
  
In fact, I don't think you need to know *anything*, as long as you can stay alive somehow.   
  
So they're right: you don't need to know math, and you can get by for your entire life just fine without it.   
  
But a few things I've learned recently might surprise you:

1. Math is a lot easier to pick up after you know how to program. In fact, if you're a halfway decent programmer, you'll find it's almost a snap.
2. They teach math all wrong in school. Way, WAY wrong. If you teach yourself math the right way, you'll learn faster, remember it longer, and it'll be much more valuable to you as a programmer.
3. Knowing even a *little* of the right kinds of math can enable you do write some pretty interesting programs that would otherwise be too hard. In other words, math is something you can pick up a little at a time, whenever you have free time.
4. Nobody knows all of math, not even the best mathematicians. The field is constantly expanding, as people invent new formalisms to solve their own problems. And with any given math problem, just like in programming, there's more than one way to do it. You can pick the one you like best.
5. Math is... ummm, please don't tell anyone I said this; I'll never get invited to another party as long as I live. But math, well... I'd better whisper this, so listen up: *(it's actually kinda fun.)*

**The Math You Learned (And Forgot)**  
  
Here's the math I learned in school, as far as I can remember:  
  
*Grade School*: Numbers, Counting, Arithmetic, Pre-Algebra ("story problems")  
  
*High School*: Algebra, Geometry, Advanced Algebra, Trigonometry, Pre-Calculus (conics and limits)  
  
*College*: Differential and Integral Calculus, Differential Equations, Linear Algebra, Probability and Statistics, Discrete Math  
  
How'd they come up with that particular list for high school, anyway? It's more or less the same courses in most U.S. high schools. I think it's very similar in other countries, too, except that their students have finished the list by the time they're nine years old. (Americans really kick butt at monster-truck competitions, though, so it's not a total loss.)  
  
Algebra? Sure. No question. You need that. And a basic understanding of Cartesian geometry, too. Those are useful, and you can learn everything you need to know in a few months, give or take. But the rest of them? I think an introduction to the basics might be useful, but spending a whole semester or year on them seems ridiculous.  
  
I'm guessing the list was designed to prepare students for science and engineering professions. The math courses they teach in and high school don't help ready you for a career in programming, and the simple fact is that the number of programming jobs is rapidly outpacing the demand for all other engineering roles.  
  
And even if you're planning on being a scientist or an engineer, I've found it's much easier to learn and appreciate geometry and trig after you understand what exactly math *is* — where it came from, where it's going, what it's for. No need to dive right into memorizing geometric proofs and trigonometric identities. But that's exactly what high schools have you do.   
  
So the list's no good anymore. Schools are teaching us the wrong math, and they're teaching it the wrong way. It's no wonder programmers think they don't need any math: most of the math we learned isn't helping us.  
  
**The Math They Didn't Teach You**  
  
The math computer scientists use regularly, in real life, has very little overlap with the list above. For one thing, most of the math you learn in grade school and high school is continuous: that is, math on the real numbers. For computer scientists, 95% or more of the interesting math is discrete: i.e., math on the integers.   
  
I'm going to talk in a future blog about some key differences between computer science, software engineering, programming, hacking, and other oft-confused disciplines. I got the basic framework for these (upcoming) insights in no small part from Richard Gabriel's [Patterns Of Software](http://www.amazon.com/o/asin/0195121236), so if you absolutely can't wait, go read that. It's a good book.   
  
For now, though, don't let the term "computer scientist" worry you. It sounds intimidating, but math isn't the exclusive purview of computer scientists; you can learn it all by yourself as a closet hacker, and be just as good (or better) at it than they are. Your background as a programmer will help keep you focused on the practical side of things.   
  
The math we use for modeling computational problems is, by and large, math on discrete integers. This is a generalization. If you're with me on today's blog, you'll be studying a *little* more math from now on than you were planning to before today, and you'll discover places where the generalization isn't true. But by then, a short time from now, you'll be confident enough to ignore all this and teach yourself math the way *you* want to learn it.   
  
For programmers, the most useful branch of discrete math is probability theory. It's the first thing they should teach you after arithmetic, in grade school. What's probability theory, you ask? Why, it's *counting*. How many ways are there to make a Full House in poker? Or a Royal Flush? Whenever you think of a question that starts with "how many ways..." or "what are the odds...", it's a probability question. And as it happens (what are the odds?), it all just turns out to be "simple" counting. It starts with flipping a coin and goes from there. It's definitely the first thing they should teach you in grade school after you learn Basic Calculator Usage.   
  
I still have my [discrete math textbook](http://www.amazon.com/o/asin/0072930330) from college. It's a bit heavyweight for a third-grader (maybe), but it does cover a *lot* of the math we use in "everyday" computer science and computer engineering.   
  
Oddly enough, my professor didn't tell me what it was for. Or I didn't hear. Or something. So I didn't pay very close attention: just enough to pass the course and forget this hateful topic forever, because I didn't think it had anything to do with programming. That happened in quite a few of my comp sci courses in college, maybe as many as 25% of them. Poor me! I had to figure out what was important on my own, later, the hard way.  
  
I think it would be nice if every math course spent a full week just introducing you to the subject, in the most fun way possible, so you know why the heck you're learning it. Heck, that's probably true for every course.  
  
Aside from probability and discrete math, there are a few other branches of mathematics that are potentially quite useful to programmers, and they usually don't teach them in school, unless you're a math minor. This list includes:

* **Statistics**, some of which is covered in my discrete math book, but it's really a discipline of its own. A pretty important one, too, but hopefully it needs no introduction.
* **Algebra** and **Linear Algebra** (i.e., matrices). They should teach Linear Algebra immediately after algebra. It's pretty easy, and it's amazingly useful in all sorts of domains, including machine learning.
* **Mathematical Logic**. I have a really cool [totally unreadable book](http://www.amazon.com/o/asin/0486425339) on the subject by Stephen Kleene, the inventor of the Kleene closure and, as far as I know, Kleenex. Don't read that one. I swear I've tried 20 times, and never made it past chapter 2. If anyone has a recommendation for a better introduction to this field, *please* post a comment. It's obviously important stuff, though.
* **Information Theory** and [**Kolmogorov Complexity**](http://www.amazon.com/o/asin/0387948686). Weird, eh? I bet *none* of your high schools taught either of those. They're both pretty new. Information theory is (veeery roughly) about data compression, and Kolmogorov Complexity is (also roughly) about algorithmic complexity. I.e., how small you can you make it, how long will it take, how elegant can the program or data structure be, things like that. They're both fun, interesting and useful.

There are others, of course, and some of the fields overlap. But it just goes to show: the math that you'll find useful is pretty different from the math your school thought would be useful.   
  
What about calculus? Everyone teaches it, so it must be important, right?   
  
Well, calculus is actually pretty easy. Before I learned it, it sounded like one of the hardest things in the universe, right up there with quantum mechanics. Quantum mechanics is still beyond me, but calculus is nothing. After I realized programmers can learn math quickly, I picked up my [Calculus textbook](http://www.amazon.com/o/asin/0471381578) and got through the entire thing in about a month, reading for an hour an evening.   
  
Calculus is all about continuums — rates of change, areas under curves, volumes of solids. Useful stuff, but the exact details involve a lot of memorization and a lot of tedium that you don't normally need as a programmer. It's better to know the overall concepts and techniques, and go look up the details when you need them.  
  
Geometry, trigonometry, differentiation, integration, conic sections, differential equations, and their multidimensional and multivariate versions — these all have important applications. It's just that you don't need to know them right this second. So it probably wasn't a great idea to make you spend years and years doing proofs and exercises with them, was it? If you're going to spend that much time studying math, it ought to be on topics that will remain relevant to you for life.   
  
**The Right Way To Learn Math**  
  
The right way to learn math is breadth-first, not depth-first. You need to survey the space, learn the names of things, figure out what's what.   
  
To put this in perspective, think about long division. Raise your hand if you can do long division on paper, right now. Hands? Anyone? I didn't think so.   
  
I went back and looked at the long-division algorithm they teach in grade school, and damn if it isn't annoyingly complicated. It's deterministic, sure, but you *never* have to do it by hand, because it's easier to find a calculator, even if you're stuck on a desert island without electricity. You'll still have a calculator in your watch, or your dental filling, or something.   
  
Why do they even teach it to you? Why do we feel vaguely guilty if we can't remember how to do it? It's not as if we *need* to know it anymore. And besides, if your life were on the line, you know you could perform long division of any arbitrarily large numbers. Imagine you're imprisoned in some slimy 3rd-world dungeon, and the dictator there won't let you out until you've computed 219308862/103503391. How would you do it? Well, easy. You'd start subtracting the denominator from the numerator, keeping a counter, until you couldn't subtract it anymore, and that'd be the remainder. If pressed, you could figure out a way to continue using repeated subtraction to estimate the remainder as decimal number (in this case, 0.1185678219, or so my Emacs M-x calc tells me. Close enough!)   
  
You could figure it out because you know that division is just repeated subtraction. The intuitive notion of *division* is deeply ingrained now.   
  
The right way to learn math is to ignore the actual algorithms and proofs, for the most part, and to start by learning a little bit about all the techniques: their names, what they're useful for, approximately how they're computed, how long they've been around, (sometimes) who invented them, what their limitations are, and what they're related to. Think of it as a Liberal Arts degree in mathematics.   
  
Why? Because the first step to applying mathematics is *problem identification*. If you have a problem to solve, and you have no idea where to start, it could take you a long time to figure it out. But if you know it's a differentiation problem, or a convex optimization problem, or a boolean logic problem, then you at least know where to start looking for the solution.   
  
There are lots and *lots* of mathematical techniques and entire sub-disciplines out there now. If you don't know what combinatorics is, not even the first clue, then you're not very likely to be able to recognize problems for which the solution is found in combinatorics, are you?   
  
But that's actually great news, because it's easier to read about the field and learn the names of everything than it is to learn the actual algorithms and methods for modeling and computing the results. In school they teach you the Chain Rule, and you can memorize the formula and apply it on exams, but how many students really know what it "means"? So they're not going to be able to know to apply the formula when they run across a chain-rule problem in the wild. Ironically, it's easier to know what it is than to memorize and apply the formula. The chain rule is just how to take the derivative of "chained" functions — meaning, function x() calls function g(), and you want the derivative of x(g()). Well, programmers know all about functions; we use them every day, so it's much easier to imagine the problem now than it was back in school.   
  
Which is why I think they're teaching math wrong. They're doing it wrong in several ways. They're focusing on specializations that aren't proving empirically to be useful to most high-school graduates, and they're teaching those specializations backwards. You should learn how to count, and how to program, before you learn how to take derivatives and perform integration.   
  
I think the best way to start learning math is to spend 15 to 30 minutes a day surfing in Wikipedia. It's filled with articles about thousands of little branches of mathematics. You start with pretty much any article that seems interesting (e.g. [String theory](http://en.wikipedia.org/wiki/String_theory), say, or the [Fourier transform](http://en.wikipedia.org/wiki/Fourier_transform), or [Tensors](http://en.wikipedia.org/wiki/Tensor), anything that strikes your fancy.) Start reading. If there's something you don't understand, click the link and read about it. Do this recursively until you get bored or tired.   
  
Doing this will give you amazing perspective on mathematics, after a few months. You'll start seeing patterns — for instance, it seems that just about every branch of mathematics that involves a single variable has a more complicated multivariate version, and the multivariate version is almost always represented by matrices of linear equations. At least for applied math. So Linear Algebra will gradually bump its way up your list, until you feel compelled to learn how it actually works, and you'll download a PDF or buy a book, and you'll figure out enough to make you happy for a while.   
  
With the Wikipedia approach, you'll also quickly find your way to the [Foundations of Mathematics](http://en.wikipedia.org/wiki/Foundations_of_mathematics), the Rome to which all math roads lead. Math is almost always about formalizing our "common sense" about some domain, so that we can deduce and/or prove new things about that domain. Metamathematics is the fascinating study of what the limits are on math itself: the intrinsic capabilities of our formal models, proofs, axiomatic systems, and representations of rules, information, and computation.   
  
One great thing that soon falls by the wayside is notation. Mathematical notation is the biggest turn-off to outsiders. Even if you're familiar with summations, integrals, polynomials, exponents, etc., if you see a thick nest of them your inclination is probably to skip right over that sucker as one atomic operation.   
  
However, by surveying math, trying to figure out what problems people have been trying to solve (and which of these might actually prove useful to you someday), you'll start seeing patterns in the notation, and it'll stop being so alien-looking. For instance, a summation sign (capital-sigma) or product sign (capital-pi) will look scary at first, even if you know the basics. But if you're a programmer, you'll soon realize it's just a loop: one that sums values, one that multiplies them. Integration is just a summation over a continuous section of a curve, so that won't stay scary for very long, either.   
  
Once you're comfortable with the many branches of math, and the many different forms of notation, you're well on your way to knowing a lot of useful math. Because it won't be scary anymore, and next time you see a math problem, it'll jump right out at you. "Hey," you'll think, "I *recognize* that. That's a multiplication sign!"  
  
And then you should pull out the calculator. It might be a very fancy calculator such as R, Matlab, Mathematica, or a even C library for support vector machines. But almost all useful math is heavily automatable, so you might as well get some automated servants to help you with it.   
  
**When Are Exercises Useful?**  
  
After a year of doing part-time hobbyist catch-up math, you're going to be able to do a lot more math in your head, even if you never touch a pencil to a paper. For instance, you'll see polynomials all the time, so eventually you'll pick up on the arithmetic of polynomials by osmosis. Same with logarithms, roots, transcendentals, and other fundamental mathematical representations that appear nearly everywhere.   
  
I'm still getting a feel for how many exercises I want to work through by hand. I'm finding that I like to be able to follow explanations (proofs) using a kind of "plausibility test" — for instance, if I see someone dividing two polynomials, I kinda know what form the result should take, and if their result looks more or less right, then I'll take their word for it. But if I see the explanation doing something that I've never heard of, or that seems wrong or impossible, then I'll dig in some more.   
  
That's a lot like reading programming-language source code, isn't it? You don't need to hand-simulate the entire program state as you read someone's code; if you know what approximate shape the computation will take, you can simply check that their result makes sense. E.g. if the result should be a list, and they're returning a scalar, maybe you should dig in a little more. But normally you can scan source code almost at the speed you'd read English text (sometimes just as fast), and you'll feel confident that you understand the overall shape and that you'll probably spot any truly egregious errors.   
  
I think that's how mathematically-inclined people (mathematicians and hobbyists) read math papers, or any old papers containing a lot of math. They do the same sort of sanity checks you'd do when reading code, but no more, unless they're intent on shooting the author down.   
  
With that said, I still occasionally do math exercises. If something comes up again and again (like algebra and linear algebra), then I'll start doing some exercises to make sure I really understand it.   
  
But I'd stress this: don't let exercises put you off the math. If an exercise (or even a particular article or chapter) is starting to bore you, *move on*. Jump around as much as you need to. Let your intuition guide you. You'll learn much, much faster doing it that way, and your confidence will grow almost every day.   
  
**How Will This Help Me?**  
  
Well, it might not — not right away. Certainly it will improve your logical reasoning ability; it's a bit like doing exercise at the gym, and your overall mental fitness will get better if you're pushing yourself a little every day.   
  
For me, I've noticed that a few domains I've always been interested in (including artificial intelligence, machine learning, natural language processing, and pattern recognition) use a lot of math. And as I've dug in more deeply, I've found that the math they use is no more difficult than the sum total of the math I learned in high school; it's just *different* math, for the most part. It's not harder. And learning it is enabling me to code (or use in my own code) neural networks, genetic algorithms, bayesian classifiers, clustering algorithms, image matching, and other nifty things that will result in cool applications I can show off to my friends.   
  
And I've gradually gotten to the point where I no longer break out in a cold sweat when someone presents me with an article containing math notation: n-choose-k, differentials, matrices, determinants, infinite series, etc. The notation is actually there to make it easier, but (like programming-language syntax) notation is always a bit tricky and daunting on first contact. Nowadays I can follow it better, and it no longer makes me feel like a plebian when I don't know it. Because I know I can figure it out.  
  
And that's a good thing.   
  
And I'll keep getting better at this. I have lots of years left, and lots of books, and articles. Sometimes I'll spend a whole weekend reading a math book, and sometimes I'll go for weeks without thinking about it even once. But like any hobby, if you simply trust that it will be interesting, and that it'll get easier with time, you can apply it as often or as little as you like and still get value out of it.   
  
[Math every day](http://www.cabochon.com/~stevey/blog-rants/blog-math-every-day.html). What a great idea that turned out to be!

POSTED BY STEVE YEGGE AT [1:06 AM](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html)

**152 COMMENTS:**

Anonymous Razvan said...

Probability theory might be the most useful area for programmers, but it is also a blurry field, difficult to learn right. You might check this book for details, and there is also an free, online draft version available.  
  
[The Logic of Science](http://www.amazon.com/gp/product/0521592712)

[3:26 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114259481285957054)

Anonymous Jules said...

I'm 16 years old now, and I live in the Netherlands. I've learned probability theory this year (the basics, I suppose). We're doing integrals + logarithms now. And we're going to start with recursive functions this month (really easy if you can program).  
  
The problem with math is, at least in the netherlands, that you're "not allowed" to like it. But math is one of the few things that will still matter in the year 4006.  
  
And it is indeed much more fun to prove that  
  
fib(n) = ((1 + 5^0.5) / 2)^n - ((1 - 5^0.5) / 2)^n / 5^0.5  
  
where:  
  
fib(n) = fib(n - 1) + fib(n - 2)  
fib(0) = 0  
fib(1) = 1  
  
than to learn 200 french words.

[7:28 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114260930049210255)

Blogger [Doug](https://www.blogger.com/profile/13613968045186520579) said...

Good stuff, I posted [a few comments](http://geekpit.blogspot.com/2006/03/math-for-programmers.html) on my blog, one of the things I mention is that I think we harm students of programming in similar ways - by teaching them depth-first (gory details of latest popular language's syntax), rather than breadth first.

[8:38 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114261352375199177)

Anonymous [hxa7241](http://www.hxa7241.org/) said...

I followed a very similar path myself, and reached the same views and conclusions.  
  
From reading technical/research papers and following references one can survey the terrain and find the important topics.  
  
My main basis has been engineering and numerical maths: the practical orientation is good. For core books I liked: 'Engineering Mathematics' – Croft, Davison, Hargreaves; and 'Numerical Recipes In C' – Press, Teukolsky, Vetterling, Flannery.

[8:56 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114261456936201649)

Anonymous Anonymous said...

Fertheluvofgahd.. please write a damn book! Take us down the road with you so we can all fix the brain damage we received during math education (in the US anyway).  
  
And by the way. Your emacs must be broke cause, unless I missed something,   
219308862/103503391 = 2.1188567821898704748716880203471  
  
-Vince  
VincePlatt AT Yahoo

[9:46 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114261761757353041)

Anonymous Anonymous said...

I think Knuth's "Concrete Mathematics" contains most of the math we'd need as programmers, and is also the most enjoyable math book I've come across. Now if only I can get around to studying it ... :)  
  
Vikas Gorur (vikas at 80x25 (dot) org)

[9:51 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114261790725583003)

Anonymous Anonymous said...

How about showcasing books as you read them and posting your thoughts on them as you go through? Keep the comments open.  
  
I desperately want to learn math--and I don't mean just memorizing shit and knowing how to substitute this number here or there to get it to work. I want true understanding. I'm interested in any sites, books, or articles out there that will get me there. Please keep your posts on math coming, but keep in mind that not all of us are as advanced!

[10:05 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114261873094475703)

Blogger [John Graham-Cumming](https://www.blogger.com/profile/07545211695594850992) said...

The book I really liked at college for logic is A. G. Hamilton's Logic for Mathematicians. I found it short and clear.  
  
http://www.amazon.com/gp/product/0521368650  
  
Pretty pricey though!  
  
John.

[10:58 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262193031855622)

Blogger [Zhi](https://www.blogger.com/profile/02543363549782232374) said...

I agree with most of the your view, but there are a few things I have quibbles with.  
  
The counting part of probability theory that you are talking about is actually call combinatorics. That is a whole field of mathematics on counting.   
  
Your approach to math is certainly from a compsci point-of-view which heavily depends on the discrete side of things. Calculus..or more generally Mathetmatical Analysis deals more with the continous stuff and it is essential to science if not other engineering.  
  
My way to math was thru number theory (long story). Not necessary any one subject that you learn in school, but it's mostly an exercise in looking for patterns in number.. Math to me is to recognize pattern in abstract space and it's a matter of strategies which help in solving problem and writing programs

[11:09 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262254121659318)

Anonymous Dan Luu said...

I liked the EFT book: it's rigorous enough and it's nice and short. I've heard that Enderton is less rigorous and more readable, making it a better introduction to logic, but EFT was good enough that I haven't had any desire to read another introductory logic book.  
  
EFT: http://www.amazon.com/gp/product/0387942580  
Enderton: http://www.amazon.com/gp/product/0122384520

[11:17 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262307433133293)

Anonymous [jasperk](http://blogs.msdn.com/jasperk) said...

I couldn't agree more. Please write a book. I was one of those CS majors who did enough to get by and now that i'm in the industry I'm actually really enjoying it and find myself reading up a lot of math/CS type of stuff. I think universities should tailor an applied-math-for software engineers type of class and teach it using the methods/content you suggest.

[11:21 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262328608511167)

Anonymous Prashant said...

Great blog and look forward to subsequent articles on this topic.I also could not agree more with your comment to start learning Math, by spending 15-30 minutes a day on Wikipedia. That's been my preferred way of revisiting and expanding on Math topics. I have always wanted to learn and absorb more of the Math out there (second your breadth first approach for that) and now that world (with Firefox and it's pull down list of search engines)is a few clicks away.

[11:41 AM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262449272685831)

Anonymous Anonymous said...

Sounds like you're advocating the "new math" that was hyped about 35 years ago, when I started school. Maybe it's time to revisit "new math".

[12:08 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262613619645648)

Anonymous Anonymous said...

At my University we get this course in our first semester of our first year, "Applicable Mathematics for Computer Scientists", pretty much comprising of what you have suggested.

[12:20 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262680460211444)

Anonymous David Brunton said...

Introduction to Mathematical Philosophy, while not a logic book per se, is way fun. I'm doing a similar regimen, and it was my first book, which in turn led to the reading regimen.

[12:23 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262703127079274)

Anonymous Anonymous said...

WOW.. i totally disagree with you.  
  
It all depends on the type of programming you are doing. If you like to do encryption programming or game programming, you NEED math.. and lots of it. However, many algorithms already exist for the in-astute

[12:25 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262712393718326)

Anonymous Anonymous said...

I enjoyed  
Discrete Mathematics and its Applications  
http://www.amazon.com/gp/product/0072930330/sr=8-1/qid=1142626982/ref=pd\_bbs\_1/102-8402190-1328907?%5Fencoding=UTF8  
  
I found it does a good job of teaching logic.  
  
-Scott

[12:25 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262713070328030)

Anonymous Bryan said...

I'm currently taking a Mathematical Logic course and we're using H.B. Enderton's A Mathematical Introduction to Logic. It's pretty good but nothing compared to the way the professor teaches it. The most fun stuff is the nonstandard analysis stuff. All those hyperreals and whatnot. In fact there's a good CC lisenced ebook on Calculus, shown using nonstandard analysis. It's available here http://www.math.wisc.edu/~keisler/calc.html

[12:26 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262719438271584)

Anonymous Mike said...

I've found this book incredibly useful as a linear algebra primer, not to mention that it applies it to something I have a great deal of interest in (3d programming).  
  
http://www.amazon.ca/exec/obidos/ASIN/1556229119/qid=1142627528/sr=8-3/ref=sr\_8\_xs\_ap\_i3\_xgl14/702-1502168-0287234

[12:34 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262765323524738)

Anonymous Anonymous said...

The 'point' of all the Geometry proofs appears to be overlooked.  
  
It isn't so much learning 1/2\*base\*height for a triangle's area. It is much more along the lines of demonstrating 'math is \_deterministic\_'. This is right in line with learning programming. If I \_know\_ A, what can I figure out from that? OK, now I \_know\_ B.  
  
Geometry is just the most interesting place to learn proofs. Writing formal proofs of Trig, for instance - while \_learning\_ trig -> eeeewww.

[12:39 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262798525983136)

Anonymous Anonymous said...

This is the dumbest thing that I have read in a long time. This guy argues that he doesn't use several topics that are in the current curriculum, so therefore they shouldn't be there. He then argues for his own curriculum, which is meaningful only for computer programmers.  
  
Here's a news flash guy: not everyone goes on to become a computer programmer. Personally, my training is as a physicist, and use almost none of the discrete mathematics nonsense that is on your curriculum. Instead, I use mathematics over the reals and complex numbers, those annoying trig formulas that get you down, and a host of calculus techniques that aren't even on your list.  
  
Just because you don't use the math doesn't mean it's useless, and it sure as hell doesn't mean that the curriculum is designed wrong.

[12:41 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262807337966365)

Anonymous Anonymous said...

What about the lambda calculus?

[12:54 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262884084805577)

Blogger [the doubter](https://www.blogger.com/profile/02891185532388964486) said...

No discussion about math would be complete without mention of http://mathworld.wolfram.com. You can literally spend all day there.  
  
Great post by the way!

[12:55 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262893018380585)

Blogger [Jim M.](https://www.blogger.com/profile/11973246104339771669) said...

So the lesson of this article is : "learn what you need to know to do what you want to do."  
  
Easily said. Harder to know at age 14 what you want to do with your life.

[12:57 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262906314816977)

Anonymous Anonymous said...

Take a look at Michael Mitzenmacher's "Probability and Computing". It covers randomized algorithms, Chernoff bounds, Shannon's Law, stationary ergodic processes, and all of the really interesting stuff from a computational perspective. It's challenging stuff, but it's a tremendously well written book.  
  
\*This\* is the book I wish existed when I was learning how to program and how to analyze complexity. Everything I do nowadays is designed to drag NP-complete problems into the realm of the tractable, by any means possible. Having a firm grasp of the mathematics underlying both natural and computational processes is invaluable.  
  
Somebody else already mentioned it, but Mathworld (mathworld.wolfram.com) and Wikipedia are both great for understanding and connecting mathematical formalisms with their related concepts.

[1:03 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262938874332378)

Anonymous Christian Anderson said...

If you're a computer programmer [John von Neumann and the Origins of Modern Computing](http://www.amazon.com/exec/obidos/redirect?link_code=ur2&tag=christianande-20&camp=1789&creative=9325&path=http%3A%2F%2Fwww.amazon.com%2Fgp%2Fexplorer%2F0262011212%2F1%2Fref%3Dpd_sexpl) is one of the best Math books I could recommend to you. I think Amazon sells it cheapest but check your college bookstore. I got mine used for only $38. It was required reading for one of the computer science classes.

[1:05 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262950116487205)

Anonymous [Mark H](http://www.ieee.uow.edu.au/) said...

I am an engineering, and i must say that even though i use, discrete math to model functions for digital logic, by far and wide i use trig and differential calculus much more, as modeling changing systems is much more important to me, also the trig identities i would stress are very important, how else would i be able to use fouriers series etc etc.   
  
I think if you did some programming especially in the areas of geomotry like a game engine you would probibly find yourself working with these functions alot more.

[1:10 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114262982330770061)

Anonymous Anonymous said...

Very well put! I used to hate math, then I started playing with Flash and become interested in math as art, it really is beautiful. Also reading Bart Kosko's books were very intriguing, bridging, discussing philosophy and art...for the first time I finally understood some of it.  
  
I wish they would have talked about some of this stuff when I was in high school, I would have paid much more attention. When you start understanding the history and the actual problems that are solved and more importantly what mathematics attempts to explain...oh my...I would have been a super geek in school.

[1:13 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263003884868190)

Anonymous Anonymous said...

I think you may have given youself a skewed picture of mathematics. You left out the two most active fields in modern mathematics, linear algebra and complex analysis. These, especially linear algebra, are vital to any programming beyond the utility level. Linear algebra is the language of 3d graphics, OpenGL is practically written as linear algebra functions. Linear algebra and complex analysis together are what allow for jpeg compression and mp3 compression, by way of their child, wavelet analysis. Also, probablility is not truly a part of discrete math. The introduction that ll of us get in our first class in stats is certainly, mostly, discrete. This is because until you have had the traditional calculus cycle (Calc 1: limits and derivative, calc 2: methods of integration, calc 3: multivariable, calc 4, intro to diff. eq.) or some variation, you won't have the tools to understand the continuous part. Integration is the continuous analog of summation. I like what you had to say, I think that math teaching is screwed up in America. I think that people do need to learn arithmetic first, simply because it is an easy place to start and is vital to get by in life. What comes next is tough.

[1:16 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263020649357035)

Anonymous [awwaiid](http://thelackthereof.org/) said...

I would put teaching/learning of prime-numbers up as one of the firstmost things to do. I'd like it to be taught before simplification of fractions, for example -- if you break the numerator and denominator into their prime factorization then simplifying the fraction becomes a trivial matter of removing what is in both numbers.  
  
The point is to realize that the prime numbers are the atoms of integers. With this fundamental idea many areas of whole-number mathematics become transparent.

[1:18 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263033827942975)

Anonymous Anonymous said...

Just wanted to say great insight. I didn't realize, but despite my teachers trying to teach me the way teachers usually do, I went ahead and learned the way you described we should learn. I think the only problem is in the testing of knowledge retention. (especially with less resources to do one on one testing) I used to perform quite well at solving problems and anything 'open book'. But was, and am still notoriously terrible at memorization. Consequently I performed low, and my teachers and professors used to say, they knew I knew the stuff, but couldn't prove it. As much as it would be wonderful if people could learn for its own sake, sadly we demand accountability more than actual learning it would seem.

[1:23 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263063396400570)

Anonymous Anonymous said...

I'm in the same boat right now: going back through old college texts to brush up on my math, which has gotten very rusty.  
  
I initially hit a wall in re-learning all this stuff, and now I see that it's because I'm trying to learn it the way it was originally taught to me in college.  
  
Your way sounds much better (and a lot more fun).  
  
Thanks!

[1:35 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263135288134942)

Blogger [Jose](https://www.blogger.com/profile/01705797379482524591) said...

Great Post.

[1:36 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263139584534736)

Anonymous Anonymous said...

the doubter: I think Wikipedia has vastly surpassed Mathworld in terms of usefulness and quantity of information.  
  
I find myself (as a math PhD student) using Wikipedia far more than Mathworld these days.

[1:44 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263189673988576)

Anonymous Anonymous said...

Were you serious about Stephen Kleene inventing Kleenex? I don't think he's quite old enough for that. Kleenex was originally developed during World War I as a gas mask filter during WW I (see [this PDF](http://www.pde.state.pa.us/a_and_t/lib/a_and_t/ItPaystobeNoseyOregon.pdf)).

[1:48 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263208685465145)

Anonymous Anonymous said...

Nice post.  
  
I wanted to follow up on this though:  
  
>> If you teach yourself math the right way, you'll learn faster, remember it longer  
  
Okay, I have a six year old daughter just about to embark on the long trial of math education and wanted to ask if you could point me to any resources that you consider teach math more effectively. I agree with you regarding the formal curiculumn and I spent lots of time unlearning my math class material so if there's a better way I'm definitely game to try. At least to save my kids the same torture.  
  
Thanks!

[1:49 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263218799519528)

Anonymous [Jon](http://reasonable-heretic.net/) said...

This is brilliant.  
  
I live in Canada, and our math curriculum is screwed.  
  
to begin with, i always think of myself as good at math. I have been doing a program called Kumon (it started in Japan, it sums up to being 25 minutes of math a day, everyday, 365 days a year) and i had a HUGE start on it. It is quite logically structured, and makes everything quite easy to learn.the only arguement i would make about it is that is disallows calculators, however i think that for basic arithmetic one should be able to do it mentally anyways. To make a long story short I can do linear algebra in my head while the teacher starts writing the first step on the board.  
I did Kumon for 7 years. thats roughly 2500 days. I have worked as an instructor at it for the last 2 and a half year.   
  
today i dropped math C30 IB.  
  
for those who dont know of the program, the International Baccalauriete is a worldwide andvanced program. dont do it. warn friends, kids, people on the street. DONT GO INTO IB.   
  
It is not worth it. I wont get into that here.   
  
what i want to get on is:  
Trig. trigonometry is nuts. the way it is taught makes it violate all the other teachings up to this point. There is so much that would be more usefull. Someone needs to remake the curriculems.

[2:24 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263424026506719)

Anonymous [hacker not cracker](http://www.hackernotcracker.com/) said...

I totally agree. Math is not that big of a prerequisite for programming. Programming is more of a prerequisite of math. I get better in my high school and college math because i could see similarities in programming.

[2:37 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263503203282956)

Anonymous Anonymous said...

AWESOME POST! I almost never leave comments on blogs ... but this one really hit home. Keep up the good work and I'll be checking back!

[2:44 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263545598535152)

Anonymous Anonymous said...

Yes I agree. Please give us links and resources that makes learning math, \*fun\*. I love the Wikipedia suggestion. I'm going to start doing this now.  
  
Any more? Where does a newbie start?  
  
  
  
Please save the children from torture ->  
  
>>Okay, I have a six year old daughter just about to embark on the long trial of math education and wanted to ask if you could point me to any resources that you consider teach math more effectively. I agree with you regarding the formal curiculumn and I spent lots of time unlearning my math class material so if there's a better way I'm definitely game to try. At least to save my kids the same torture.

[2:53 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263602788635599)

Anonymous [Vicki](http://www.cfcl.com/vlb) said...

Wonderful article.  
  
I had some trouble with math up through the 8th grade. So, when most of my peers were taking Algebra 1, I took "Arithmetic II". And then I "got it" and things were better. And yes, in many ways math was fun.  
  
But before that there were a lot of tears. And I don't use much of what I learned in my daily life as a programmewr/tech writer. (Then again, MOST of what is taught in high school, and college for that matter, has no bearing on Real Life After School)  
  
By the way, my hand is up. I can do long division on paper, right now

[2:56 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263619379333487)

Anonymous Anonymous said...

Goedel, Escher and Bach - An eternal golden braid. Douglas Hofstadter, 1979.  
  
Not sure I would call it a textbook though...

[2:59 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263636088551989)

Anonymous [Vicki](http://www.cfcl.com/vlb) said...

Re: "Godel, Escher, Bach"  
see also  
http://www.cabochon.com/~stevey/blog-rants/godel-escher-blog.html

[3:05 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263673251413112)

Anonymous Anonymous said...

If you want to learn Quantum Mechanics/Electrodynamics, pick up a very slim book called "QED" by Richard Feynman. It was made from a series of four lectures that lays out the whole deal -- it even explains the rainbow reflections on CDs.  
  
I've found that the best explanations often come from the original inventor. Einstein's "miracle year" papers are short, clear & brilliant.

[3:13 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263719339968550)

Anonymous Anonymous said...

I don't know anyone, even the highschool flunkout, that can't do long division.   
  
Anyone else think this guy is out to lunch on this topic?

[3:13 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263723949846158)

Blogger [MathCogIdiocy](https://www.blogger.com/profile/03947814210371350192) said...

The answer to the question of who determines the math curriculum in the US is the NCTM (National Council of Teachers of Mathematics). They have a set of standards for each grade level. States use these standards in whole or part when determining what the specific state's curriculum will include.   
  
You can find information on NCTM standards here - http://www.nctm.org/standards/overview.htm.

[3:40 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114263882886123331)

Anonymous Anonymous said...

I agree and yet disagree. It depends upon the programming being done. As one person mentioned, graphics requires a lot of math if you program from the ground up, such as the the transforms between 3d to 2d planes.  
  
On the other hand, if programming say a CRM program, not a lot of math is needed to track a customers tables and information within a database.  
  
So, again, it depends upon what is being programmed. I have programmed in both fields and can see the huge difference in programming with relation to math skills needed.

[4:14 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264084734345179)

Anonymous [Jesse](http://www.jessebickel.com/) said...

I'm not sure about this. One part of me agrees with breadth first (it's how I prefer to operate) but another part of me says mastering a few things is healthy and good.  
  
Basically this guy is telling us to learn on our own, like we should be doing anyway. The reason math is not interesting to people is not any failure of the teachers, it is a failure of people to open their eyes and think of learning as something they do outside the classroom. You don't have to tell me to use Wikipedia to find interesting things.  
  
People respond to this post with "oh great! write a book because I agree with you and I'll buy it!" when the whole point of the article is to just go find and read some things that are interesting and useful to you.  
  
So on the whole, this is a mixed bag.  
  
I Agree: go learn on your own things that interest you  
I Disagree: the school system is screwing us up

[4:16 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264101698906971)

Anonymous Devin said...

Don't forget to read Godel, Escher, Bach.  
  
http://www.amazon.com/gp/product/0465026567/qid=1142641394/sr=2-1/ref=pd\_bbs\_b\_2\_1/102-7588289-3509704?s=books&v=glance&n=283155

[4:25 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264150218661611)

Blogger [freaka](https://www.blogger.com/profile/10010652107501552142) said...

hey hey. Kleene closure is simple, all you're doing is making a of words from an alphabet.

[4:45 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264271876702117)

Anonymous Anonymous said...

it really is nonsensical to teach a division algorithm to pupils if they forget it shortly after the test on the subject. there has to be a page in the textbook that just lists the individual steps of the algorithm, because there are people who are good at rote memorisation and will actually remember it forever and be able to use it if the need ever arises. but how probable is that? it's still nifty to have people who can do math on paper around, in case someone drops an h-bomb on your country, all electronic calculators are destroyed and you have to deal with the logistics of distributing the now scarce supplies.  
  
but it's much more important to explain why and how the algorithm works. firstly, because there are other "learning types"; people who can't remember a list of seemingly arbitrary computational steps easily, but will be able to perform the calculation effortlessly once they 'get' it. and secondly, and that's the important part, because this provides you with actual insight into the subject matter and builds problem solving skills that you can make good use of every single day in your life.  
  
i don't recall the lessons in school when i was told how to do long division, or which trigonometric functions to apply to which triangles, or which combinatorial functions to apply to which problems. but if i had to do any of this right now, i could easily figure it out, because it's nothing more complicated than middle school math.  
  
maybe it's true that "math is a lot easier to pick up after you know how to program." (maybe it's also true that programming is a lot easier to pick up after you know your math.) but suggesting in the same writeup to use the simplest possible (and mind-numbingly tedious) algorithm to perform a division doesn't *quite* drive that point home. if you're a computer scientist, or even just a comp-sci student in the first year, doing this would be downright idiotic. you're supposed to be able to make up a better algorithm on the spot.  
  
there sure are some things in that entry that ring true to me. however, i am definitely not taking someone serious who labels **primary** school material "annoyingly complicated". something every 10-year-old is expected to understand simply isn't "annoyingly complicated", even if there are better ways to teach it.

[4:54 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264329405649046)

Blogger [Doug](https://www.blogger.com/profile/07773477347537321256) said...

You might want to read up in the areas of the learning sciences, instructional design, cognitive psychology, and related fields. See for example the book "How People Learn" and an instructional design book called "Understanding by Design".  
  
There has already been a tremendous amount written on problem-based learning (PBL), anchored instruction, enactive learning, and other techniques which have all been applied to mathematics education. See for example the Jasper Woodbury CDROM series.  
  
A similar theme in all of them is you learn in context, while solving a real-world problem. It's called situated cognition, or situated learning.

[5:08 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264413676544549)

Anonymous Anonymous said...

If you want fun mathematical logic, you can't go wron with Raymond Smullyan. His books typically start with a series of fascinating logic problems, with a story to tie them together, and then segue into an extended and enjoyable study of some branch of logic and foundations that almost always ends up in a discussion of some of Godel's work from a new direction each time. I particularly suggest "The Lady or the Tiger", "Satan, Cantor and Infinity", and my favorite "To Mock a Mockingbird" in which he introduces a great deal of combinatory logic (aka lambda calculus aka Lisp) in the guise of a birdwatching expedition in a series of strange forests. Incidentally, this is especially appropriate because the main founders of the subject were Moses Schonfinkel - the name means "pretty bird" - and Haskell Curry, an avid birdwatcher. You might also get some mileage from "Infinity and the Mind" and the mathematical fiction "White Light" by Rudy Rucker. There is also an old Dover book called "How to Calculate Quickly" that teaches actual useful number sense based ways of doing four banger arithmetic that, practiced well, will make you largely independent of a calculator and greatly improve your arithmetic intuition.

[5:21 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264491234444172)

Anonymous Anonymous said...

It really seems that in the past calculus and differential equations were at the heart of various engineering edeavors. Now, however, it seems that probability and discrete math are becoming more important, espeically given the new interdisciplinary mash-ups like 'genetic algorithms/genetic programming' (combining evolutionary biology with algorithm design), various social-insect based search methods (ACO, PSO - another biology meets engineering mashup), bioinformatics, etc. Probability and statistics play a big role in genetics and biology in general and now we're seeing biology and engineering combining.

[5:59 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264716752879495)

Anonymous [eric](http://www.ericdfields.net/) said...

The amount of comments here are simply staggaring! Just wanted to say that i found your article to be a good read and even inspiring; I'm a coder in the XHTML/CSS sense (I do it professionally), but I am enrolled in a programming class where we do JavaScript, PHP, and ActionScript. When it comes to doing projects for the class, I have big ideas but very little time to do them, and when I approach something, I just get frustrated. The more interesting stuff I do in ActionScript involves trig ("The only useful math," my professor once said), and the more I read about programming properties of springs and such, the more I wish i understood what i was copying and pasting off the net. Makes me realize how much they go hand and hand, and hearing this stuff from an adult who is still learning both programming skills and math makes me feel a lot better. Thanks!

[6:14 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264809720406928)

Blogger [Axl](https://www.blogger.com/profile/13991784089891272983) said...

<http://www.amazon.com/gp/product/1558606793/sr=8-1/qid=1142648877/ref=sr_1_1/104-2348453-7316769?%5Fencoding=UTF8>

[6:30 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264901829950236)

Blogger [Mario De Roma](https://www.blogger.com/profile/05388502689425924148) said...

That whole post is so positivist it makes me shiver.  
  
I mean: yeah, you're right, grown ups learn stuff the other way round (was that "breadth-first" you said?) and that's a fact I like someone else but me noticed. Thanks.  
  
Then:  
1) One STILL cannot get away with saying "I read it on wikipedia" (get some public official endorsement for it and that will probably change - then: get it and probably wikipedia will lose editors)  
2) There will be times in life you really has no damn calculator handy or you can't trust one, (hey, TCPA is coming!)  
3) School isn't aimed at churning out programmers, and in case, I'd wonder.   
  
About that then, maybe at a certain point it the future it will; and with everybody programming (lawyers and medics alike) old CS graduates will end up being underspecialized.

[6:37 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264942461926385)

Blogger [mfwic](https://www.blogger.com/profile/02660674498965462917) said...

I was in EE grad school in the late 90's and I took a semester of discrete-time signal-processing. The prof mentioned a push in academia to teach "DSP first". I liked the idea as soon as I heard it.  
  
In discrete-time math integrals are summations and differentials are subtraction problems. It makes understanding the mechanics of calculus much easier. Discrete-time math is a great way to intro calculus to anyone. After you understand the fundamentals you can, if you choose, take continuous-time calculus. Of course anyone that wants to, should be able to go straight to continuous-time. But those majors that don't really need advanced math shouldn't be stuck with only one choice for calculus.  
  
  
[mfwic](http://mfwic.blogspot.com/)

[6:42 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264974347719459)

Anonymous Anonymous said...

Excellent article. It's strange that I've started teaching myself math the same as you. It starts with wikipedia, and it usually starts on a less math related subject, such as turing machines, string theory, heim theory, quantum computers, etc.   
  
I think what I agree with most in your article is that math should be taught in a broad format first. It would be much easier and more enjoyable if the concepts are taught first instead of being drowned in exercises and ROTE memorizations. That's what references are for.

[6:42 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114264977297179682)

Anonymous Anonymous said...

The best that can be said about this article is that it's a muddleheaded morass of untrue truisms; the worst that could be said is that we're dealing with a crackpot gibbering drivel.  
Calculus is not "easy." If you think you disagree, try solving a few simple integral equations. After you recover from that debacle, you can have fun with the calculus of variations; and when you pick yourself up the floor bleeding and backbroken from that one, try a little conformal mapping. Integrals on the Argand plane operate differently than in the domain of the reals, as Cauchy's Theorem assures us. But of course the author of this article knows neither Cauchy's Theorem, nor conformal mapping, nor any of the other basic elements of calculus he falsely and glibly misrepresents as "easy."  
In most real-world programming tasks, probability theory proves as worthless as it is pointless. The vast majority of programming involves traversing lists and doing various kinds of sorts. For these tasks, graph theory and queuing theory remain indispensible. Naturally, these areas of math never made it onto the author's list of things allegedly "important to know." Linear algebra has essentially no connection with real programming in the real world, unless you need to solve systems of simultaneous equations, and in most cases there are a lot simpler ways of getting to asolution that works in the real world than solving sets of simultaneous linear equations. The Monte Carlo method is one example (and you don't need to know probability to implement Monte Carlo, you only need a decent psuedorandom number algorithm) and there are many more.  
So many other false claims and outlandish pronouncements litter this article (like turds in a punchbowl) that by the time you've read 2/3 of the way through, you realize it's all unbelieveable tripe.  
Arrogance has always been the besetting sin of programmers. Unfortunately, unlike engineers who build bridges or physicists who build rockets, a programmer whose design fails can always just blame the machine -- "bad compiler," "not enough memory," and so on.  
Math is indeed taught badly in America, but 99% of it, you never need to know. I have yet to meet any electrical eningeer who ever needed to know how to do a Laplace Transform to get a circuit working, I have yet to meet a CS graduate who ever had to know the details of Cohen's 1964 completeness proof to get a program working, and I have yet to meet a programmer who claimed that [fill in the blank: advanced math or exotic pgrogramming language] is actually "simple and easy to learn" who wasn't an ignorant incompetent fool.

[6:57 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114265065764636243)

Anonymous Anonymous said...

Interesting stuff. I fully agree with self directed learning of any subject. But I think your distance from your youth has blinded you to the real complexity of math, particularly from the perspective of a student encountering the theory for the first time. Yes, gifted students can and will advance faster than the teacher be teaching. However, you must remember that they are teaching to ALL the students in the class. Also, the point of the depth first teaching is to get students to learn one thing well before encountering more difficult and abstract mathematical concepts. It's easy enough to state that integration is summation of a continuous function. It's another to teach kids to recognize that velocity is the integral of acceleration and that all these functions can be related graphically and are fundamentally connected mathematically.  
  
I think it is safe to say that your ability to move easily from one subject to another owes much to the teachers that spent countless hours drilling into our heads the fundamentals of mathematics. You can learn algebra without calculus, but not vice versa.

[7:42 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114265334322766167)

Anonymous Anonymous said...

The classic logic book is Boolos, Burgess, and Jeffrey, *Computability and logic*  
  
http://www.amazon.com/gp/product/0521007585/sr=8-1/qid=1142654323/ref=pd\_bbs\_1/102-5472329-8864945?%5Fencoding=UTF8

[8:02 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114265452207643023)

Anonymous Anonymous said...

I think it was a divine act that led me to this blog. Recently I've had a huge urge to learn math (I'm very mathematically ignorant), but I've had no clue as to where I should even start. So thanks for the great blog. I'm adding it to my favorites.

[8:05 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114265473907053858)

Anonymous czar said...

Here are a couple of books that I would highly recommend for anyone who wants to learn math in the context of some interesting problems :  
  
The Art and Craft of Problem Solving by Paul Zeitz  
Problem Solving Through Problems by Loren C Larson  
anything by Titu Andreesu

[9:41 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114266047058527837)

Anonymous [Jerome](http://www.slinkydink.be/) said...

Hi there. Just need to extend a huge thanks to you. Just into my first semester of comp sci and its been a while since ive done any maths. Sometimes i'll sit around and hear the sorts of things some people are talking about during class and i feel like i really dont belong. Its just a big confidence booster for someone like to me to read something like this. In the end, it makes me feel a little better. Just because they can do all of that fancy math I still get higher grades in programming.  
  
I too shall be keeping an eye on your blog.  
  
Jerome

[9:49 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114266097355751768)

Anonymous Anonymous said...

I agree with the general tenor of this post. I have long railed against the way math is taught; it seems calculated to turn most people off. I also think that teaching children that mathematics is the study of patterns--any pattern--and then giving them some understanding of things they feel viscerally (e.g. rates of change: a fast ball hurts more than a slow one, etc.) would do much to remove the "mystery." Exactly how to approach teaching math in this way is something I have been working on for a few years. I hope to teach my own children plenty of mathematics before they ever narrow their visions to mathematics as simply manipulation of abstract symbols.  
  
One reservation though: It seems to me that your approach to mathematics ( the just-in-time method?) leads to only surface understanding that can be dangerous--literally deadly. Probability and statistics, is an area, for example, where these errors can be especially egregious since for some it seems deceptively simple to some. Subtleties that come from deeper knowledge are lost.  
  
I have found in my experience, my co-workers and in my hiring, that I prefer programmers/developers whose undergraduate degrees were not in Computer Science. Most Comp Sci-only undergraduate programs seem to leave the graduate with knowlege of a few languages, some algorithms, and precious little else. No domain knowledge of any kind in solving real world problems (even basic laboratory exercises would help); and little intuive feel for when a result makes no sense.   
  
In fact, I believe that Computer Science, as generally taught now, should not be an undergraduate degree: it should be a graduate degree. Mathematics, Chemistry, Engineering, Physics, Biology and a few other fields seem to make a better foundation for programmers in my experience.  
  
It boggles the mind that there are CS majors out there who have not had Abstract Logic, one of the very foundations of the discipline. I won't bother listing other areas that are lacking.   
  
By way of full disclosure I should state that my degrees are in Mathematics, Electrical Engineering, and Political Science. I have many years of experience in different industries and I am doing some graduate work in Molecular Biology in between doing a bunch for hiring for a startup.   
  
And has anyone pointed out that Computer science IS mathematics?

[9:54 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114266128045046972)

Anonymous Anonymous said...

Ummmm, the reason why programmers might not find math useful is because they don't know it.  
  
I started studying graduate level probability and then, all of the sudden, respect started pouring my way.  
  
It's sad, but true, that US companies have to send work to other countries because the US workforce doesn't do math.  
  
A good read is:  
  
"On the Creulty Of Really Teaching Computing Science"  
  
http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1036.PDF  
  
by Dykstra.

[9:56 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114266141874982549)

Anonymous Anonymous said...

opps, look like the URL was cut off:  
  
/ewd10xx/EWD1036.PDF  
  
is the ending

[9:59 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114266155025118054)

Anonymous Anonymous said...

The biggest thing for me about math and computer programing is doing proofs. Writing a proof is a lot like writing a program. The main difference is that other people have to be able to understand your proof - the same can not be said about your program.

[10:24 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114266307995028851)

Anonymous [Jason](http://sluagh.warped.com/) said...

I totally agree that math is taught wrong, especially at the high school level. I'm a neuroscientist in a hospital and while I rarely ever see calculus, I'm up to my neck in statistics on a nearly daily basis. Anyone in any health-related field knows that the math they use most of the time is statistics. An understanding of statistics is essential to understand any sort of scientific study, especially those regarding human health. Most people never learn statistics beyond mean, median and mode, but are expected to make decisions about their health that involve higher level statistics and probability. And it's not just medicine where statistics are important, but business, government, and many more areas.  
  
I think it would be much more beneficial for students in high school to get a good foundation in statistics, then calculus or pre-calculus.

[10:58 PM, MARCH 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114266511539356020)

Anonymous spacenookie said...

The modern math curriculum was in place by the 1930's. At that time companies employed rooms full of "computers", people with math training, to do all the things that today would be done with a desk calculator or spreadsheet. Engineering firms employed roomfuls of engineering graduates who hand-drafted blueprints and solved calculus equations. Math education hasn't changed much since then.  
  
I agree with much of what the article says. If you can frame your problem in mathematical terms you can find somebody who will solve it for you.  
  
I sometimes remark that mathematicians become programmers (mathematica, matlab) while programmers, for the most part, forget math.

[12:27 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114267045896920379)

Anonymous Anonymous said...

Forgive me for what might seem like a double post but I felt compelled to correct (and add to) the initial post; it was was written in haste before I had to go out. Perhaps it's the mathematician in me.  
  
I agree with the general tenor of the original article as far as the teaching of mathematics is concerned. I have long railed against the way math is taught; it seems calculated to turn most people off. I also think that teaching children that mathematics is the study of patterns--any pattern--and then giving them some understanding of things they feel viscerally (e.g. rates of change: a fast ball hurts more than a slow one, etc.) would do much to remove the "mystery." Exactly how to approach teaching math in this way is something I have been working on for a few years. I hope to teach my own children (when I have them) plenty of mathematics before they ever narrow their visions to mathematics as simply manipulation of abstract symbols.  
  
A few reservations though: It seems to me that your approach to mathematics ( the just-in-time method?) leads to only surface understanding that can be dangerous--literally deadly. Probability and statistics, is a vast area that encompasses Thermodynamics and much of Quantum Mechanics, for example. Errors can be especially egregious here since it seems deceptively simple to some. Poorly designed clinical trials are but one example. Subtleties that come from deeper knowledge are lost in this "simplicity."  
  
I have found in my experience--my past co-workers and in my hiring--that I prefer programmers/developers whose undergraduate degrees were not in Computer Science. Most Comp Sci-only undergraduate programs seem to leave the graduate with knowlege of a few languages, some algorithms, and precious little else. No domain knowledge of any kind in solving real world problems (even basic laboratory exercises would help); and little intuitive feel for when a result makes no sense. These graduates are often too narrow.   
  
In fact, I believe that Computer Science, as generally taught now, should not be an undergraduate degree--it should be an advanced degree. Mathematics, Chemistry, Engineering, Physics, Biology and a few other fields seem to make a better foundation for programmers in my experience.  
  
It boggles the mind that there are CS majors out there who have not had Abstract Logic, one of the very foundations of the discipline. I won't bother listing other areas that are lacking. "CS majors think that they don't need math?" Huh?!  
  
A couple of posts prior to this one explain in some more detail the fallacy of this shallow approach: the "learning-just-enought-to-get-by" philosophy. Unfortunately it shows in the results of the programmers who come from the background reflected in the article. Some things are hard to learn and no amount of simplification will attain the necessary depth. Real airplanes will crash, patients will get an overdose of X-rays, real human beings can actually die.  
  
By way of full disclosure I should state that my degrees are in Mathematics, Electrical Engineering, and Political Science. I have many years of experience in different industries as an engineer and a programmer among other things; and I am doing some graduate work in Molecular Biology in between doing a bunch of hiring for a startup. Though, likely I have some bias, I think these opinions come from hard-won experience.  
  
And has anyone pointed out that Computer science IS mathematics?

[3:26 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114268121340356458)

Anonymous Anonymous said...

Forgive me for what might seem like a double post but I felt compelled to correct (and add to) the initial post; it was was written in haste before I had to go out. Perhaps it's the mathematician in me.  
  
I agree with the general tenor of the original article as far as the teaching of mathematics is concerned. I have long railed against the way math is taught; it seems calculated to turn most people off. I also think that teaching children that mathematics is the study of patterns--any pattern--and then giving them some understanding of things they feel viscerally (e.g. rates of change: a fast ball hurts more than a slow one, etc.) would do much to remove the "mystery." Exactly how to approach teaching math in this way is something I have been working on for a few years. I hope to teach my own children (when I have them) plenty of mathematics before they ever narrow their visions to mathematics as simply manipulation of abstract symbols.  
  
A few reservations though: It seems to me that your approach to mathematics ( the just-in-time method?) leads to only surface understanding that can be dangerous--literally deadly. Probability and statistics, is a vast area that encompasses Thermodynamics and much of Quantum Mechanics, for example. Errors can be especially egregious here since it seems deceptively simple to some. Poorly designed clinical trials are but one example. Subtleties that come from deeper knowledge are lost in this "simplicity."  
  
I have found in my experience--my past co-workers and in my hiring--that I prefer programmers/developers whose undergraduate degrees were not in Computer Science. Most Comp Sci-only undergraduate programs seem to leave the graduate with knowlege of a few languages, some algorithms, and precious little else. No domain knowledge of any kind in solving real world problems (even basic laboratory exercises would help); and little intuitive feel for when a result makes no sense. These graduates are often too narrow.   
  
In fact, I believe that Computer Science, as generally taught now, should not be an undergraduate degree--it should be an advanced degree. Mathematics, Chemistry, Engineering, Physics, Biology and a few other fields seem to make a better foundation for programmers in my experience.  
  
It boggles the mind that there are CS majors out there who have not had Abstract Logic, one of the very foundations of the discipline. I won't bother listing other areas that are lacking. "CS majors think that they don't need math?" Huh?!  
  
A couple of posts prior to this one explain in some more detail the fallacy of this shallow approach: the "learning-just-enought-to-get-by" philosophy. Unfortunately it shows in the results of the programmers who come from the background reflected in the article. Some things are hard to learn and no amount of simplification will attain the necessary depth. Real airplanes will crash, patients will get an overdose of X-rays, real human beings can actually die.  
  
By way of full disclosure I should state that my degrees are in Mathematics, Electrical Engineering, and Political Science. I have many years of experience in different industries as an engineer and a programmer among other things; and I am doing some graduate work in Molecular Biology in between doing a bunch of hiring for a startup. Though, likely I have some bias, I think these opinions come from hard-won experience.  
  
And has anyone pointed out that Computer science IS mathematics?

[3:28 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114268128261507552)

Anonymous [lgatto](http://www.slashhome.be/wp/) said...

Thank you for this nice article. I particularely appreciate (i) the detached and relax way to approach and study math you describe, (ii) your general tips and (iii) the list of books your mention.  
I will watch out for more of your blogs.

[3:33 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114268163988636557)

Anonymous Anonymous said...

You can follow this link to get a good free ebook about mathematical logic, written by Stefan Bilaniuk:  
  
http://euclid.trentu.ca/math/sb/pcml/pcml-16.pdf

[4:05 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114268353158189967)

Anonymous [Dave Rosen](http://www.futuretrack5.com/) said...

You should write a book man.   
  
I enjoyed this not just for the Math angle but also that you went about teaching yourself something new. I think when a lot of people leave school they just doing continue to expand there brain and slowly you become a little stagnent.   
  
No matter what the subject if you are feeding your brain some fuel its sure going to help you with programming or whatever.  
  
Really great work.

[4:20 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114268443779427672)

Anonymous Kirk said...

Count, me, a baby boomer, as one who has always had problems with higher math. Oh, and did I say that I was the product of 'new math' taught during the early sixties? For nearly three years I got the 'new math' shoved down my throat and never learned it. That little experiment in education cost me a whole lot of time and money.   
  
Can I use math? Yes, but without the proper foundations it came harder and harder the further I went into mathematics. Real world you have reference books easily accessible and if you know what information you need you then look it up. In the world of academics you memorize, memorize, and memorize. Does that produce mathematicians? Nope, that produces parrots who squawk back the correct answer when asked the right question. Fineman (SP) discovered that when he went to Brazil and saw their colleges.   
  
All in all this was a good read.

[6:12 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114269112320582732)

Anonymous Anonymous said...

I stopped reading when I got to the point where the author states he thinks Kolmogorov complexity is more important than calculus. This article is a joke is so many regards there's not even any point to picking it apart.  
  
I'm not surprised people want to believe it, though.

[6:29 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114269216629543092)

Anonymous Anonymous said...

I do math for a living and I like to find people that realize that math is useful outside of the math they learn in school.  
  
For a programmer calculus is close to useless unless you are doing computer graphics, but I think that some of the more discrete stuff must come up in just about every program ever written. I don't recommend ignoring long division because there are generalizations of this idea called 'integral domains.' where this is important when you need to find the greatest common divisor (important in cryptography applications).  
  
For logic I recommend book that I used in philosophy [The Logic Book](http://www.amazon.com/gp/product/0072401893/sr=8-3/qid=1142695662/ref=pd_bbs_3/104-6009773-7592734?%5Fencoding=UTF8). Most discrete math texts also cover logic.

[7:31 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114269589553208200)

Blogger [Ron Bannon](https://www.blogger.com/profile/09483895832430777135) said...

I've written a short book on stochastic modeling that might be suitable for beginning programmers. Email  
  
ron .dot. bannon .at. gmail .dot. com  
  
with "stochastic models" in the subject line if you'd  
like a draft copy --- I'm basically looking for comments.

[8:38 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114269989374667873)

Anonymous Anonymous said...

I was having this very conversation with my boyfriend this morning. I'm a programmer and he's a lawyer. Neither of us is the least bit scientific, and we just passed enough maths to get to university. But we don't remember \_anything\_ from school maths beyond the age of about 9. The whole thing was so badly taught... we were both just saying that we know the terms like trigonomometry, logarithm, etc... but have no idea how they fitted together or how we should use them. In fact, I find that after spending so many years in school not understanding maths and not even being told **in what practical circumstances I would ever be able to use any particular piece of maths**, I even have a kind of math phobia or dyslexia now, where I can barely even think of mathematical symbols without freezing... But I realised years ago that the key to kids understanding this stuff was for them to learn the practical applications first, so they could then see just how knowing maths could save them from having to 're-invent the wheel' in solving real problems. It's amazing that I've been able to work as computer technician and then as a programmer given that I have such a mental block on maths.  
  
In fact, in my more cynical moments, I see that for the majority of kids school just serves as some kind of day-time prison - designed to keep them off the streets and out of trouble while their parents work. I can't think of anything of any value that I learned in school (and I went on to take 5 degrees). How much of a waste of time is it for all those other kids.

[9:32 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114270313587309313)

Anonymous Anonymous said...

I don't know how you managed to skip Number Theory. A solid basic number theory book (Kochy comes to mind right off hand) is incredibly useful for programmers. Using your example of dividing large numbers, some basic congruence relations can make that trivial to extend to far more elaborate situations, as well as giving you a really strong set of 'tricks', like casting out 9s to check sums, efficient parity checks, etc etc.

[10:20 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114270604711954716)

Anonymous [Szili](http://www.flickr.com/photos/szili/) said...

Thank you! I've gained back my motivation to learn for my comprehensive exam. Especially for that darned Kolmogorov complexity… ;)

[11:29 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114271018906937269)

Anonymous Anonymous said...

Well Stevey, I'm happy for you that you're learning more math and enjoying it, but I think your article was rather weak.  
  
So you (one person) didn't learn much math in high school or university, because \*they\* taught it wrong. First, your new-found expertise in statistics should tell you a sample size of one isn't too impressive. Second, you might be learning more because you're much more motivated, since you're picking what you want to learn. Would you have picked anything as a teenager?  
  
I think the most important point (i.e. one of the ones I agree with, ha!) is that it is important to motivate students to learn something, and to engage their intuition. I remember learning vector calculus, and finding it difficult, and then learning the physics (electromagnetism) that gave birth to the math, and suddenly understanding it all (okay, sadly not all, but much more!).   
  
Perhaps it's another a bad generalization, but I think people learn much better when they can imagine what's happening and apply their intuition, rather than just pushing symbols around (which is also interesting later, in computer science).  
  
What I personally like about funky math is it seems to stretch my brain in whole new ways, and I end up coming out with a better understanding of how the whole universe fits together. And when you see concepts from different areas coming together (linear algebra for multi-variable calculus, tensors of inertia looking like covariance matrices from statistics) it's pretty cool.  
  
And yes, I can do long division on paper. Hell, I can take a square root, but I'm a geek who likes math (and uses it daily, in my programming work -- mainly 3D stuff). I'm also lucky in that I'm good at it, and had a number of good teachers.  
  
So yes, students should learn more linear algebra and statistics, and yes, it's possible to learn math as you get older, but no to most of the rest of your "I have solved the world's math learning problems" post.

[11:49 AM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114271136825682783)

Anonymous Ewan said...

I Disagree with you saying maths teaching is wrong. Maths needs to be taught basics first, can you remember what the first Maths you were taught in school? To Count next to add/subtract then that multiplication is adding several times and division is how many times you can subtract one number from another. By the time you get to colledge level the idea is that you start to specialise, here in england for a standard maths A-level you can specialis in stats, discrete, mechanics. The problem is most children don't see how certain basic pieces of maths fit in later on and get board learning them before they get on to the interesting/fun Maths (I'm 16 and I would be there if it wasn't for som of my maths teachers).  
By the way while I'm doing my A-levels I look up maths on both mathworld and wikipedia and share it with my maths teachers, it makes interesting discusions about maths I want to look into. I'm also taught certain elements of maths before anyone else is this way.

[1:03 PM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114271583950034291)

Anonymous [Patrick Hall](http://blogamundo.net/dev) said...

Really liked your post.   
  
Here's a great book that's been challenging me for a couple years:  
  
[David MacKay: Information Theory, Inference, and Learning Algorithms](http://www.inference.phy.cam.ac.uk/mackay/itila/)   
  
The whole thing's online, too.

[1:20 PM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114271682286722745)

Blogger [Amit Kulkarni](https://www.blogger.com/profile/03625975005884279324) said...

I am glad I found your post through osnews.com!! I am seriously thinking of applying some of the stuff in our field.  
  
Look forward to more of your math ramblings

[2:40 PM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114272164837635893)

Anonymous Anonymous said...

Quantum is easy.  
  
Quantum I: Solve INT(f~(x) \* f(x) dx)=1 where f~(x) is the complex conjugate of f(x).  
  
Quantum II: Solve H\*Psi=E\*Psi where E is the energy, Psi is the particle(s)'s space-time equation, and H is the hamiltonian (like you learn in mechanics) written in terms of operators (p -> -i\* hbar \* d/dx).

[2:52 PM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114272236393614395)

Blogger [-=DD32=-](https://www.blogger.com/profile/13102532721672162825) said...

Great blog.  
  
First up its my first year of university, And right up until the last 2 years of high school i actually knew WTF i was doing in maths, then came the last 2 years at a different school and i was immediatly going WTH is this stuff?  
  
I was being tought to do who knows what and not being able to understnat WHAT it was used for, shoveling equations down my throat and telling me it finds x when something else is useless of i cant reconise when the equation would actually be useful.  
  
So now come university, And i'm doing discrete mathematics as a first unit, rather easy to understand straight off the bat unlike many others who thought they were exelent at maths the last 2 years..  
..Exept, Long division is something i was NEVER tought to do, thats right, i was never tought how to do it, i really couldnt work it out.. it was assumed that everyone would know how to do it when the past 7 years i had been told "No, dont worry about it, just use the calculator, you'll allways have one handle, and if you dont, theres subtraction" Something i agree with..  
So now has come the time that i need to know long division -- Binary Division -- I'm still not 100% sure on how to do it... but supprisingly, guess how i got to my current understanding of it?  
I Wrote a PHP function to do exactly that, infact whenever i have a maths concept i cant understand right now, i write a PHP function/program to do it manually without relying on all the builtin maths functions(Ok, FLOOR() and ROUNDUP() to round to next higher or lower number, but thats only because humans can do it in an instant whereby computers need much more steps), but you know what? Its actually teaching me how to do it in my head, i can SEE the logic behind it, i can SEE where road divides and theres a choice of performing 2 different tasks which are part of the same concept.  
  
And one again i can actually follow and excel in mathematics once again.  
It may not be a Computer Science degree(Actually its Bach of Information Technology -- but no-one comment on that) But still requires abit of knowledge..  
  
Other thing i've noticed? Those who try to do functional programming and repeditive calling of functions(technical term escapes me right now) usually have absolutly NO idea how to solve a complex function.. Once tought how to do the function maths correct, and THEN teaching the programming they get it every time..  
  
There are some maths that needs to be tought before Programming, but theres alot more maths which should be tought after the basic techniques of programming.. that may sound stupid to a non-computer person, but how often will you see in a school someone who enjoys using computers -- particuallly programming -- suceed and understand the work alot easier? How often are the ones who have absolutly no maths knowledge the ones who look at programming and can absolutly not understand a word of it.. even if it is plain simple what it does, if someone knows no maths, they wouldnt be able to pick it up usually...  
  
Great Post, and if blogger has a RSS feed you just scored yourself another reader.  
  
D

[3:00 PM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114272285305504472)

Anonymous Anonymous said...

Thanks, nice article.

[4:17 PM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114272747243256026)

Anonymous Anonymous said...

It was nice to come across this blog, for until now I did not really know anybody else who is a   
math-hobbyist. I personally believe that Math stretches my understanding of things and I am able to   
think in a much more logical way when I am doing math on a regular basis. And yes Steve, I use ONLY   
pen and paper for doing my math-any math, mostly discrete though. Pen and paper gets me the feel which I find   
lacking in reading from a site or having my sums done by a calculator-it lets me have a grassroot level view of  
things at work (and believe me, thats how most people do Math in India. Or maybe, thats how they used to , when   
I was in school). When it was a part of my syllabus, I never really paid much attention to it. But years later, when   
I am a software engineer, dabbling in Java,Oracle,Unix,PERL, I find thats something I miss from those days.---Abhishek Ghose

[7:06 PM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114273760610723126)

Anonymous Anonymous said...

No drink involved?  
  
Blasphemy!

[8:13 PM, MARCH 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114274163202026772)

Blogger [Dr MindHacker](https://www.blogger.com/profile/12451307530806818434) said...

a moron wrote:  
  
Anonymous said...  
  
"This is the dumbest thing that I have read in a long time. This guy argues that he doesn't use several topics that are in the current curriculum, so therefore they shouldn't be there. He then argues for his own curriculum, which is meaningful only for computer programmers...." blah blah blah  
  
Try reading the title of the article before leaping out of the jungle of your emotions and into an intellectual discussion. This dude is talking about math for \*programmers\*, so yes, he is recommending a curriculum accordingly, and I find it a very thoughtful one.

[7:43 AM, MARCH 19, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114278300204069391)

Anonymous Anonymous said...

You need calculus to learn probability!! Probability is not counting, it is much more, and you need at least calculus.  
  
Combinatorics is a very slippery field, not as easy as you seem to be implying.  
  
Matrices have traditionally been taught in high school. I learned them there, as well as basic combinations and permutations, Pascal's triangle, binomial expansions, gaussian elimination, Kramer's rule, etc. in a working class high school in Richmond, CA. (Known for it poor education.)  
  
Matrices are an example of a vector space, by no means equivalent to Linear Algebra.   
  
Linear Algebra is the single most important math subject in undergraduate education. You begin to do real math, serious abstraction, and it is applicable to everything under the sun.  
  
One of the reasons that computer science has progressed at the rate it has is the use of abstraction. It is everywhere in computer science. Abstraction is also at the heart of learning mathematics. You are focusing on applications and missing the real point: power thinking. That is what enables you to do really great applications.  
  
You are doing the breadth, but haven't really gotten to the depth yet, at least not in this article.  
Keep working. No matter your route to mathematics, it will pay handsomely in the end if you keep it up.  
  
Some of what you are experiencing is mathematical readiness. You are ready, and you weren't earlier. But, I, too would change the learning of mathematics. I would eliminate calculators so that the experience of "how things go" is being learned all along. Students who learn without calculators "get it" in a way that those who use them just don't.  
  
I would emphasize building things and geometry very early on. Make it as important as arithmetic. It should be as physical as possible.   
  
Have teachers that love math teach it! Most of our teachers couldn't even do math.  
  
One more thing, long division is the first really interesting algorithm. It halts, and it is easy to see that it does. What gave you your first understanding of algorithms?   
  
Hope to see what you have to say in another five years, after you have done this for awhile.

[10:01 AM, MARCH 19, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114279131921280980)

Anonymous Paul said...

I am a mathematics teacher at a community college. I have never taught math at the high school level. I hope this won't sound too definsive, or too cynical.  
  
You definitely have many valid points. Statistics and Probability (including counting methods) is very useful and can be taught the high school level. I would replace high school geometry with stats if I were in charge.   
  
I do have two primary differences with what you are saying based on what I try to accomplish as a teacher, and what you want out of math as a student and professional.  
  
First, you are (you openly state) looking at it from a programers point of view. I will reitterate what someone else posted. Calculus (with trigonometry)is the most powerful mathematical tool there is, because most situations are modeled with functions which are continuous. Thus, mathematics in the public school is single minded in it's approach to get a student to a point where they can succede in a calculus class. The only side trip traditionally taken is for Geometry. I, myself,would advocate taking Geometry out of the high school list. The main reason for that class is to teach a student to understand and be able to produce a logical proof. A skill most students don't need until after they have mastered the basic skills of mathematics, and I am including Calculus as a "basic skill".   
  
The second problem that I have with your analysis is that you, and most of your readers, are intellectually far, far above the average high school and even college student. You mention that because you are a programer you know what a function is, and what it does. I will tell you that College Algebra is the first college level math course that a non-science oriented (someon who did not get through at least Algebra II in high school) will take. College Algebra is roughly defined as the study of functions and their graphs. Most of my students leave my college algebra class, despite my best efforts to continuously talk about the concept, without truly understand the concept of a continuous function. Because they are not capable of understanding it.  
  
I will agree that this leaves high school and college mathematics with a problem. There was a day not long ago that anyone who recieved a college degree had to have at least one semester of calculus, even if they where an art major. The notion that you will ever use this in real life only applied to Calculus, Trig, and again I will include Stats. Algebra, by itself, is nearly useless. The so-called "application" problems that are supposed to mimic real life situations are ridiculus. The only purpose for learning Algebra is to be able to do Calculus. And while it is true to a degree that the concepts in Calculus are not all that hard to understand themselves, the algebra involved in successfully navigating anything other than polynomials is generally what defeats most Calculus students. My favorite saying in that class is "It's not the Calculus that is going to get you here, it's the algebra."  
  
Then we started trying to educate larger and larger segments of the population. People who would never dreamed of getting a degree 30 years ago are now giving it a shot. But most of them are not capable of mastering Calculus. Many of them cannot fight there way past basic algebra. They cannot master the algebraic manipulations and/or the concepts necessary to be successful. But, educators insist that they prove themselves at least somewhat worthy by forcing them to have some mathematics.  
  
So, I have given up on justifying math other than stating that what they are accomplishing here is to prove to someone (the people who run the education system) that they can perform long and tedious procedures without making mistakes at least 70% of the time. It's about the best you can hope for in terms of the general population.  
  
I have daydreamed about a solution. I think what should be done is to restructure math so that a student would take a one semester course in polynomial algebra, where they mastered thos basic skills. No roots, no rational expressions (algebraic expressions which look like a fraction), just polynomials. Many more students could handle that. Then they would take "Polynomial Calculus". Polynomials handle a majority of the real life situations, and you could demonstrate the power of Calculus. Math and science majors would then need to take more clases which would teach them to deal with more complicated algebraic functions, then apply the Calculus concepts to those functions.

[11:22 AM, MARCH 19, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114279617463465985)

Anonymous Anonymous said...

A great book on mathematical logic, or modern symbolic logic as I have known it, is "Logic : Techniques of Formal Reasoning" by Kalish, Montague, and Mar. I enjoyed your article, thanks!  
~David.

[12:32 PM, MARCH 19, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114280036512955083)

Anonymous Anonymous said...

There is much in this blog with which I agree. I like the idea of breadth first, for instance. But there is an underlying assumption about the role of education that I disagree with. The assumption is that education is primarily about preparing us for our vocations.   
  
The truth is that in democracies the chief goal of education is to produce educated people. Two properties seem obvious in this regard. One is that we share common ideas and backgrounds that bind us together culturally. Speaking the same languages (English, Algebra, Geometry, Philosophy, Economics..) and understanding the same facts about the nature of the world and the human role in it (Physics, Chemistry, History, Literature, Biology...)  
  
The second goal is closely related; that educated people can talk to each other about inportant issues that lie outside their own field of expertise and help inform decisions made inside the field. This is especially crucial in political discourse.  
  
Frequently one finds that what one learns in school has direct application to one's vocation. And when this is true, education provides economic value. Ecomomic prosperity does play a significant role in securing freedom. But even if this were not true, education would still be both the first and the last line of defense of a democracy. (The argument is too long to present here; read Montesquieu.http://www.constitution.org/cm/sol-02.htm)  
  
Let's take geometry as an example of a common language. In a strict sense, geometry is almost useless as vocational training - except to mathematics students who go on to teach the subject. But in a broader sense, the construct of geometry is deductive reasoning. And the sort of reasoning used here is crucial in a host of disciplines. It may be courtroom law that draws on the mental disciplines and processes inculcated by geometry.  
  
I was educated as an engineer and I believe that the continuous math models need much more exposure, not less. For these models pervade all physical sciences, medicine, economics, and biology. I frequently see places where applications of continuous models -even fairly informal cobbled together ones - might improve decisions in fields of sociology and public policy.  
  
Computer science is a vitally important field. One of the most interesting courses I took in the field was on algorithms and computability. It was my first exposure to the mathematical concept of proof by contrapositive. ( Assume the opposite of what you are trying to prove and show that this leads to a contradiction ) This, it turns out is a powerful technique in a host of fields. Safety analysts of all sorts use an adaptation that posits "If I were designing the least safe system, what would I do." Then "how does this resemble the system I have designed." It turns out that this kind of analysis could be done for most services - telephone computer tech support comes to mind first - to much benefit.  
  
So really, each field has a number of tools and techniques that can be brought to bear on a problem. And each solution is improved when the right set of tools have been used. The whole purpose of breadth is to escape the old problem that "when all you have is a hammer, everything looks like a nail."

[12:59 PM, MARCH 19, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114280194978474158)

Anonymous Anonymous said...

1. First there was Archimedes who invented most of the (applied) math  
...  
Then there was Felix Klein and other german matematitians who invented "fus(/z)ionism"  
Then Hardy wrote an appology and said that math should teach You to think so he feel purest math is in fact most applied   
Then von Neumann again after Archimedes knew or maybe invented all of the math we call applied now  
Then french bourbakists   
Then i myself tried to do the right connection ... and lost really bad ...;-)  
with time i learnet to forgive people who don't want to see ... a really expensive exercise in christian ethique   
Now it's Your turn  
  
Every math - here is the point - writen by great mind is good, useful and teaches not only how to think but some great tricks and most important ethique and to make difference between science and publicity (last become a huge problem for IT ,most people here prefere to forget that IT exist becouse of von Neumann's math genius and that there are some promises that digital world gane never keep so it is better not giving them).....  
Another important experience i think is that it is better to start with math You understand for me it was easy to understand intuitionism , forcing and non standard analysis than dedekind cuts different than couch(y/i?) reals if not axiom of choice is valid ....  
my (incommplete)list of very best autors is :  
Archimedes  
Felix Klein   
Hardy  
von Neumann  
Bourbaki   
Abraham Robinson  
Paul Cohen   
Dana Scott ,R. Karp , Rabin   
  
and Feynman!!!!not exactly a math ...from math point of view read last ... will see how good he was .I started at 13 and it was the first i read so i have a giant unfair advantage later ... and read the real thing lectures on physics , computing ... not QED  
last word on logic Kleene is one of the best and most original in field but well his view is specific and ... i prefer to be Feynman's slave than Kleene's ... there is good handbook of logic editet by Jon Barwise too  
  
All that are old books and citizens of richer places in the world can and should try to find shorter and better way but this is part of my own way long time ago and it is what you can find if you are an ex-communist citizen (in russian too) any way all these autors are extremely good and if not pressed to much for time (if you can find for reading Kleene you have enough ;-)  
I write teribly bad in english but i try to think in math (when i think at all )and it is not a very great problem for me

[3:48 PM, MARCH 19, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114281212585206409)

Blogger [Abhinav](https://www.blogger.com/profile/04428860297324962487) said...

It would be nice if you could post a list of books that you are currently reading/have read/will read

[12:34 AM, MARCH 20, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114284367586258598)

Blogger [Chad](https://www.blogger.com/profile/04407685449467940830) said...

I appreciate what you are saying. I'm coming at this from the angle of, "how am I going to teach my kids math?". My wife and I homeschool our kids, so we have the luxury of teaching math in whatever manner we prefer. I've read some compelling arguments for starting math relatively late, say 10 years old. Prior to 10, teach the "grammer" of math. The vocabulary of it, if you will. This would be the "breadth" approach that you describe so well. Then, when they are well versed in the "language" of math, teach the application of a particular discipline.   
  
It's sort of akin to the old analogy of filling buckets vs. lighting fires. Much of state run education today has become a process of filling "buckets". Fill each kid with these facts at this age and by this age they'll now this, this, and this. The lighting fires approach teaches a wide variety of subjects, developing a love of learning and discovery that lasts a lifetime.   
  
Rote memorization may make you an efficient programmer, but it won't make you especially creative. Creativity is the birthplace of great software, products, art, etc.

[6:12 AM, MARCH 20, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114286392381635602)

Blogger [trr](https://www.blogger.com/profile/05432339037788264156) said...

Someone mentioned that the name Schonfinkel means "pretty bird". That's not quite right. It means pretty finches. Schoenvogel would be "pretty bird".  
  
I enjoy reading the math articles on wikipedia and mathworld.wolfram.com and I've always liked reading books *about* math (not math textbooks), but I don't feel that I really *know* the math having read them. I read the Hofstadter book many years ago, and it really captivated me. What remains after all the years is the interestingness of self-referential and recursive things. More recently I've read books about Erdos (biographical) and the story behind proving the Riemann hypothesis. It talks about the importance of prime numbers, as someone else mentioned, but getting to the proof required a lot of non-discrete math.  
I do agree with the assertion that a lot of people are put off by the notation of math, but how to get around that? You can't do without it, can you?

[10:51 AM, MARCH 20, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114288069076801279)

Blogger [trr](https://www.blogger.com/profile/05432339037788264156) said...

I should mention: Math in school was usually not fun for me, except Geometry class. That had a lot to do with the teacher. Of course I did well enough at it, but it could have been a better experience.  
Science was more interesting than math, but as an engineer, both are necessary.  
I didn't know John von Neumann was a chemical engineer until I read it on Steve's page titled "Math Every Day". Cool!  
  
PS Steve, I lived in Kirkland, WA when I was in elementary school.

[11:17 AM, MARCH 20, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114288222077050390)

Anonymous [Vicki](http://www.cfcl.com/vlb) said...

Some comments in the following refer to maathematics and programming  
  
http://www.cs.utexas.edu/users/EWD/transcriptions/EWD13xx/EWD1308.html  
  
esp. the 2nd para from the end  
  
"Looking back I cannot fail to observe my fear of formal mathematics at the time. In 1970 I had spent more than a decade hoping and then arguing that programming would and should become a mathematical activity; I had (re)arranged the programming task so as to make it better amenable to mathematical treatment, but carefully avoided creating the required mathematics myself."  
  
Prof.dr Edsger W. Dijkstra, June 2001

[12:13 PM, MARCH 20, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114288561432300229)

Anonymous Anonymous said...

Steve, I've got a topic for you to write/rant/whatever on - how does one (e.g. @AMZN) predict time for a project with great accuracy, so that 95% of the time you can actually meet your deadlines? They're starting to treat programm{ing,ers} now as a factory assembly worker/line rather than as art + science. What is all your knowledge of math and what not worth, if you can't make your pointy haired boss look good by giving something that can cut the mustard and nail the timeframe on the head every time? I hope GOOG's treating you good (or better, I shud say, than where you were).

[1:05 PM, MARCH 20, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114288875519797274)

Anonymous Anonymous said...

I think "The Art of Computer Programming" by Donald Knuth is a fine way to spend some time. Most of programmers didn't read it, shame on them (I am such a programmer ;-), I read only some small parts from it). The main worth of the multi-volume book is its "interactiveness", it was written like a course with examples which are like a bridge between theory and practice.

[12:12 AM, MARCH 21, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114292872450467146)

Blogger [Anthony](https://www.blogger.com/profile/11331225099257806179) said...

*the simple fact is that the number of programming jobs is rapidly outpacing the demand for all other engineering roles.*  
  
No, no, it isn't. The number of actual *programming* jobs isn't very large, and has always, even in the height of the bubble, been lower than the number of real engineers. Most IT jobs are not programming, even if some of them do some incidental programming for automating routine tasks.   
  
So your advice about which sorts of math ought to be taught isn't terribly useful, even though your advice about how to teach math may be.  
  
(There's an interesting article by Joel Spolsky on what makes for a good CS education [here](http://joelonsoftware.com/articles/ThePerilsofJavaSchools.html).)

[7:45 AM, MARCH 21, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114295595789933319)

Blogger [raf.](https://www.blogger.com/profile/06676175696379502135) said...

I studied Computers in Spain, University of Oviedo. We used to joke about the discrete maths, calling them discreet maths.We didn't have the slighest idea of what they were, altough we had a subject on them.   
  
Lead by the joke, once I realised what they were, and how the math teachers were fooling us the students teaching us continuous maths. I have felt relief reading your article.  
  
There has also been brain damage for us computer engineers and programmers here in Spain.  
  
Something needs to be done (globally!) to finish the tiranny of the continuous math teachers over the discrete science.

[9:28 AM, MARCH 21, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114296210107960364)

Anonymous Anonymous said...

A nice place to learn linear algebra is a set of free video lectures of Gilbert Strang's MIT class at:  
  
http://ocw.mit.edu/OcwWeb/Mathematics/18-06Spring-2005/VideoLectures/index.htm  
  
It's a great resource for learning linear algebra, or for anyone who wants to learn how to teach an abstract subject.  
  
Actually, lots of the MIT OpenCourseWare stuff is a good resource.

[9:29 AM, MARCH 21, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114296214907547742)

Anonymous Anonymous said...

In my experience (20 years, plus) more time is spent fiddling with money and time, than any other subject. Whether it is hours worked, or compound interest, outside of pure engineering, money is a part of most calculations.  
  
Converting a million dollars into Yen will definitely take double precision, and maybe more. Even "simple" calculations can get fouled up if you fail to take precision into account.  
  
The reason money is so important, is that Most of the people hiring programmers are trying to get (or keep) more of it, and spend less of it.

[10:33 AM, MARCH 21, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114296598249121546)

Blogger [j.](https://www.blogger.com/profile/09007126819565460844) said...

If you want to learn mathematical logic, I recommend you the book "The incompleteness phenomenon". It is an excellent (and broad) introduction to the subject. That's the place I learned it from. It is well written and has lots of nice exercises.

[12:10 PM, MARCH 21, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114297180334067592)

Anonymous Anonymous said...

In your post, you asked for a good book on logic for computer scientists. I highly recommend [Logic in Computer Science](http://www.cs.bham.ac.uk/research/lics/) by Michael Huth and Mark Ryan. And you might like some stuff on logic I write at [Mathematics and Computation Blog](http://math.andrej.com/). Andrej.

[1:29 PM, MARCH 21, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114297655811161758)

Blogger [cincin21](https://www.blogger.com/profile/14020768703578298370) said...

Hey Stevey!!  
  
Awesome Blog!!  
  
I am a retired High School Math teacher ... now a private Math tutor ... an espert in Algebra I and II ... Geometry ... Trig/Pre-Cal and Calculus ...  
  
I loved what you had to say and I will be reading Wikipedia 30 minutes every day because I love to learn new things .... and it might help me understand the TV show NUMB3RS better!  
  
I have often used your "hostage" situation with my students whenever they asked me "When will I ever need this?" my response ... "well ... you might be held hostage by radical member of the Isoscelic group Al-Gebra ... and one of those memmbers might put a gun to your head and say ... 'Solve this or die!!'" My students' responses were usually "I choose death!"  
  
But I taught the 3 time Geometry repeaters ... after they took Algebra I a couple of times ... and the ESL students ... and anyone else they could dump on me.   
  
We are making kids hate Math ... and standardized testing is only making it worse (not to mention severly watering down the math we do teach).  
  
I do enjoy your blogs! Keep it up!!

[2:07 PM, MARCH 22, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114306524083455199)

Anonymous Anonymous said...

How about floating point ? Most non-engineers write very little floating point code, and understand little about precision and round-off.

[6:26 AM, MARCH 24, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114321041564569640)

Anonymous ilya said...

hi,  
  
i would say math education nearly everywhere sucks.  
  
for example, numbers and figures are inherently connected so that you can study them simultaneously - that's how mathematicians actually prove theorems about them (even so "applied" as elliptic curves cryptography) - but unless for some reason you decided to hide it you honestly have absolutely no idea what I mean. you just wasn't educated in that (and you can't get that knowledge from the list of standard undergraduate subjects of which your list is a subset)  
  
the problem is they haven't taught neither you, nor all math majors in all except very few departments what are mathematicians really doing about mathematics. those who are really going to do math, of course, eventually figure by themselves (usually in grad school)  
  
if you'd like to do something about that, you can do a high-end mathematics web meeting point, like web university or whatever.

[7:48 AM, MARCH 28, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114356092837841828)

Anonymous Anonymous said...

Alright, I recommend an introductory ebook on mathematical Logic by [Robert A. Herrmann: Logic for Everyone](http://www.serve.com/herrmann/books.htm).

[4:01 PM, APRIL 03, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114410530218512468)

Anonymous Wilson (aka CuriousMind) said...

If you want to practise math skills and programming skills at the same time, check out [Project Euler](http://www.mathschallenge.net/index.php?section=project) at [MathsChallenge.net](http://www.mathschallenge.net/).  
  
Their own description:  
  
Project Euler *is a series of challenging mathematical/computer programming problems that will require more than just mathematical insights to solve. Although mathematics will help you arrive at elegant and efficient methods, the use of a computer and programming skills will be required to solve most problems. Please be warned that the problems are challenging and you are unlikely to make much progress if you have no knowledge of programming. Having said that, this is both the perfect environment and opportunity to learn new techniques and refine your programming skills.*  
  
The problems range from the relatively trivial to quite challenging!  
  
See you there.

[10:44 AM, APRIL 12, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114486389421338695)

Anonymous Kay said...

Steve,  
  
I cannot refrain from commenting. While I find it commendable that you approach math, this definitely isn't it.  
Sorry to be blunt. Math isn't easy. No part about it. As we used to say in uni: "Math is an asshole".  
I cannot agree with the simplifications you, although I generally enjoy reading your blog.  
Math in my opinion is about structure, not necessarily about content. That's why math is important for all scientific/engineering trades.  
It's not about to be able to calculate this or that probability, nor is it about evaluating an integral or solving differential equations.   
If you want to understand math, start with the number systems. The basic axioms (Peano!) and go on from there. Learn the linear algebra involved (what is a group, what a ring? How about a field? Does it have to be commutative? What happens if it isn't?) - you'd be suprised how different linear algebra is from applying Gauss' algorithm...  
I dare say that almost no programmer 'knows' what a function is.  
Mathematical reasoning is a very important field to know. How could you prove anything without it?  
If you do not need pencil and paper you aren't doing math, you're doing calculations in your head.  
Regarding the allegedly bad math education: What's being taught isn't exactly math. It's its application to the problem domain of children. You are exposed to math in AP courses if you are lucky enough, but most are probably not.  
Being able to recite Euler's formula to directly sum up the first n natural numbers is nice. But how about a proof? Does it actually work? That's the thing that's changing your point of view of problems. Is this solution I have in mind correct? Does it even exist? What are the presumptions I am making?  
All those nifty formulas I can look up, but the way of thinking about problems I cannot find on wikipedia. But that's what mathematics is about.  
All the things you mention are important for sure. But if you do not have the knowledge of the fundamentals (proof by induction, what are my numbers, what basic properties do my fields, rings, groups have) you simply do not stand a chance to get the idea of calculus. Depth first. Breadth later. There's no shortcut around that. At least not if you are serious about mathematics.

[1:13 PM, APRIL 17, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114530483697466162)

Anonymous Anonymous said...

A few points:  
you dont need to have an excuse like being a better programmer or writing slightly different programs to learn maths - maths is the bedrock of all our understanding of the physical world, period - anyone who ponders the ubiquity of mathematics in every attempt to understand the nature of the physical world has motivation enough to study it.  
you talk about the right way to learn math - the right way is the surely the way that leads to real understanding - (whatever that is - prepare to slide down the tip of the iceberg) -   
but there is such a thing as mathematical modelling - identifying something you're facing as an instance of BLAH where BLAH is some mathematical approach already taken for you by great minds.   
if you dont "know the names" as you rightly point out, you will not even be able to categorise the problem you face in the right way, and have no hope of finding even a pointer to the solution.  
  
There is also the kind of background knowledge that takes the mystery out of design choices within programming languages say - or at least makes certain things click ( lambda calculus and lisp, category theory and haskell etc)  
  
Programmers need abstraction of the problem domain primarily - they dont \_need\_ maths as such like you say, but studying maths or physics tends to teach abstraction ( the mental act itself) better than comp sci ( despite the focus on creating language facilities FOR abstracion - its a use/mention prob)  
for a physicist you model the earth AS a sphere - sometimes as a point; a system with all its parameters is just a space with many dimensions, states become points in it etc etc ) - all this is training to abstract ( as in abstract from) detail to get to the model.  
mathematicians will idenitfy an operation and then ask whether its commutative, associative, or what ve you, by doing so they will then be able to pigeonhole it and formulate truths about it based on the structures revealed.  
  
so maths will given you the some structures to work with , but if you ask yourself - are there more structures than the ones talked about?,or are there connections between these apparently differemnt groups of strucures? - then you're entering into mathematical research itself - and that seems a different ballgame - this is the iceberg itself  
  
finally , children learning maths -   
- those who have the most understanding ( mathematicians!) should consider it an obligation to society to publish their "path to understanding" - these should be widely disseminated and chewed over by educationalists - what they had the good fortune to find, just might work for others... - a shared database of key, transformative insightful experiences

[9:25 PM, APRIL 22, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114576631926228042)

Anonymous Anonymous said...

I read through the blog a while back, and like you I've had a spotty maths career to date. Recently, I've been looking into ways to improve this, and come to realise that discrete maths is the bees-knees when it comes to relevance to computer science. Have a look, for example, at the table of contents of Rosen's book (Discrete Maths and its Applications) and try and find one chapter that doesn't sound relevant! Recursive functions, trees, graphs, Boolean algebra - it's all good, honest, useful math. Somehow, when my teachers tried to force-feed me calculus, I just never wanted to eat my vegetables. Now, older and wiser, discrete maths looks positively delicious (and calculus still isn't very appetising).  
  
Anyway, the main point of this post is to introduce you all to a FREE set of VIDEO LECTURES in discrete mathematics I've recently come across. These lectures are released under a Creative Commons license so download and go nuts! The lecturer is \*excellent\*, and has spurred an interest in mathematics in me I never knew I had. The course roughly follows the contents of Rosen's book I mentioned earlier. There are other video lectures on computer science available too. It's a shame the ArsDigita University that they came from didn't survive past it's first year. Here's a link to the main site. From there you should be able to navigate to the various courses and download the lecture video and slides.  
  
http://www.aduni.org/  
  
-- Robert

[7:58 AM, APRIL 25, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114597708418046120)

Blogger [Flux](https://www.blogger.com/profile/03471806952132395505) said...

Excellent post, as mentioned by the 121 people before me.  
  
Wow! I can read numbers!  
  
Just have to say.. you've given me hope.. lol..  
  
I fancy myself as a fairly decent programmer. I graudated from DESIGN at UCLA and have never touched a calculus class in my life, with the silly notion that I don't need math to program.  
  
  
Whoooooo boy. ....  
  
Come one quater where I did a project with some physics simulation with springs. The springs were exploding! I look up the cause. Ah. I need a Fourth Order Runge Kutta...  
  
wtf..  
  
I didn't even know what an --Integral-- is. And by looking it up, every definition I could find is completely circular: they all defined these strange math words with more strange math words! Sometimes, defined by the same word I'm looking up!  
  
At this point I realize that I am seriously crippled in my math knowledge. Every single formula I look up I demanded a programmatic or procedural approach to thinking through the problem... and all I get is gibberish mathematical equations that might as well be intercepted alien messages.  
  
Rigid body dynamics? Nope. Can't do it. Need math.  
  
Springs exploding? Nope. Can't fix it. Need math.  
  
Want to draw a NURBS curve? Nope. Don't understand what the stupid symbols mean.  
  
So yes. I will pick up a math book and must teach myself. Thank you for all the recommendations.  
  
There is hope!!!

[3:41 AM, MAY 03, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#114665286221811346)

Blogger [Amy](https://www.blogger.com/profile/09449769888474944583) said...

I love the way a friend of mine (double major, math & CS) said it: CS is math that does stuff.

[10:59 PM, JULY 30, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#115432554430112261)

Blogger [Phil](https://www.blogger.com/profile/14495944100763013607) said...

You have some good points and some, in my humble opinion, not so good points. But your opinion is skewed by the fact you're a programmer, just as my opinion is skewed by the fact I'm a mathematician.   
  
You're suggesting that Calculus and other maths in which the reals or complex numbers are used should not be focused on in relation to Discrete or Finite Mathematics.   
  
I disagree. And reason why is that in terms of how much certain subjects encompass, the fact is that the theory of continuous sets of numbers is simply more general than the discrete counterparts. Think about it. Everything from Statistics, (Linear) Algebra, DEs, Calculus, and so on require an understanding of the real/complex numbers -- this understanding is provided by Pre-Calc and later on by Calculus classes.  
  
I'm not going to comment any further, but someone wrote this:  
  
> Your approach to math is certainly  
> from a compsci point-of-view which  
> heavily depends on the discrete   
> side of things. Calculus..or more   
> generally Mathetmatical Analysis   
> deals more with the continous   
> stuff and it is essential to   
> science if not other engineering.  
  
You're much too skewed by the requirements of programming. Discrete mathematics is not as important a field as subjects concerning continuous sets -- students from business, chemistry, physics, engineering, math, biology, and so forth, require indepth knowledge of continuous systems.   
  
This is in addition to the fact that a theory concerning continuous spaces are much more general than one for discrete systems. Take the well known fourier transform.   
  
A student needs to learn about the continous fourier transform, before he or she starts implementing discrete fourier transforms in computer programs.   
  
If you're concern is how to we prepare the general populace for computer-oriented applications, then you have a point.   
  
But if your concern is how we should present the most essential mathematics in order to reach the most students, then I don't think your arguments hold much water.

[8:15 PM, AUGUST 07, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#115500691606329593)

Blogger [Steve Yegge](https://www.blogger.com/profile/14812997485690838920) said...

Fine. Let's keep calculus and pre-calculus. But I would suggest replacing trigonometry with discrete math. I just can't see anyone arguing persuasively that trig is more widely useful than probability and combinatorics. And trig isn't a prerequisite for calculus; it's completely orthogonal. Trig should be rolled into geometry at the high-school level, or even dropped.  
  
I'm not saying trig isn't useful; that'd be silly. But I was never introduced to discrete math until my first year in the CS department. I could have used it a lot sooner than that.  
  
Of course, I haven't been to high school in 25 years, so maybe they offer discrete math now.

[9:31 PM, AUGUST 07, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#115501151326569101)

Blogger [Jim](https://www.blogger.com/profile/05989166936563485250) said...

Bravo!  
  
After playing with decision trees and some neural net libraries, I realized I need to dig back into the math. The path you advocate was exactly what I was beginning to do.. Go forth and spread the gospel.

[4:08 PM, SEPTEMBER 18, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#272319747718596295)

Blogger [Maciek](https://www.blogger.com/profile/14325321343388476360) said...

Take a look at [OpenCourseWare](http://ocw.mit.edu/OcwWeb/Mathematics/index.htm) at MIT. Some of their lectures have video versions (Linear Algebra for example). Great stuff.

[11:06 AM, OCTOBER 16, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#613829242353640480)

Blogger [Danny](https://www.blogger.com/profile/04967554828613596340) said...

re. Kleenex, I've got a pile of similarly opaque books. Most readable stuff I've found on logic has been buried in AI books. John Sowa's got a neat reference page for his Knowledge Representation material at http://www.jfsowa.com/logic/math.htm  
  
(His KR book is very good but hard work).

[1:57 AM, NOVEMBER 06, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#8497321329761491869)

Blogger [critical](https://www.blogger.com/profile/16904041205458986131) said...

I have gotten good milage out of Mordechai Ben-Ari's book [Mathematical Logic For Computer Science](http://www.amazon.com/Mathematical-Computer-Science-Mordechai-Ben-Ari/dp/1852333197/sr=8-3/qid=1162820198/ref=sr_1_3/103-6964585-0646244?ie=UTF8&s=books)

[5:38 AM, NOVEMBER 06, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#6817840700535337331)

Blogger [Nick](https://www.blogger.com/profile/06955895749920431598) said...

Hi!  
I'd like to add one more author  
which had written very nice books on  
how the math and matematician work:  
George Polya  
Try his "Mathematics and Plausible Reasoning" and "Mathematical Discovery".  
  
Also his "Problem in Analyses" are awesome.  
I hope they're not out of print.  
  
Also try Alfred Renyi. I don't know  
whether his "Trilogy on Math" was translated to English, I read it in Russian translation.  
  
Good, I wish I read it when I was 13 or 14 I'm sure I'd much better in math.

[7:26 AM, NOVEMBER 06, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#6959120009538802135)

Blogger [Sammy](https://www.blogger.com/profile/00972366055112007614) said...

Asimov wrote a story about a future where math was a lost skill. Wikipedia tells me it was "A   
Feeling of Power".  
http://en.wikipedia.org/wiki/The\_Feeling\_of\_Power

[3:12 PM, NOVEMBER 06, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#2123372656406877950)

Blogger [Owen](https://www.blogger.com/profile/08801657755717528401) said...

Interesting problem I've been wrestling with is that math is a second class citizen in computing!  
  
What do I mean? Well, first off, mathematical notation is not easily available within web pages and email. MathML is a good start, but still, its a struggle.  
  
And it DOES make a difference. The typical functional forms folks use like the fib() example above are just harder to read than standard math notation.  
  
One variation that fits computers well enough is J, a computing language based on APL. Very concise and absolutely unambiguous.  
  
This leads to the second problem: there is not a lot of compelling software for maths yet, with a couple of nice exceptions. J is worth a look. R is great for statistics. GnuPlot and Octave and similar matrix packages are OK too.  
  
But wouldn't it be great to have true math notation integrated into computation?  
  
  
-- Owen  
  
Owen Densmore http://backspaces.net  
Redfish Group: http://redfish.com http://friam.org/

[9:47 AM, NOVEMBER 07, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#6262448020391279251)

Blogger [Peter Boothe](https://www.blogger.com/profile/17965824121030784289) said...

*Naive Set Theory* by Paul Halmos is fantastically readable and an excellent introduction to the subject.  
  
As far as I know there is no excellent-yet-readable book on mathematical logic.

[12:47 AM, NOVEMBER 11, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#2066184933763633098)

Blogger [Alan](https://www.blogger.com/profile/08994463605035607454) said...

Steve: Thanks for the interesting post. A great book about mathematical logic is E. C. R. Hehner's [A Practical Theory of Programming](http://www.cs.toronto.edu/~hehner/aPToP/) (freely available on the web in PDF). I took the course with him at University of Toronto in 2003; it was probably the most interesting course in my entire undergraduate CS degree.

[2:26 AM, NOVEMBER 19, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#7888072766162692772)

Blogger [logicnazi](https://www.blogger.com/profile/12167366567189261911) said...

So I'm a math graduate student (well really I'm a logic graduate student which is a math/philosophy program but my thesis is in recursion theory) and while I have great sympathy for your general tone the problem is that it simply doesn't work in practice.  
  
First though I should say I disagree totally with the just in time approach to learning math by browsing wikipedia. That's like only learning programming languages by looking at example code and never reading about data structures or good practices. Admittedly I find lecture a very bad way to learn math but actual mathematics classes, not the stupid computation stuff for non-math people, are very useful if for no other reason than getting feedback and having a book picked for you. As soon as you move beyond the very easy stuff you need consistent notation and practice problems (until you learn how to make your own) and wikipedia doesn't do either very well even though it is great for a quick definition or proof you forgot. Also I'm not convinced by your choices among fields of math.  
  
Anyway I agree that the sort of math people get in school (until you take an upper division math course in college) is designed to teach rote algorithms rather than real understanding. As a TA it makes me feel vaguely dirty to know I'm part of this process that turns off people who want to really know what's going on and might actually enjoy advanced math. Real math is fun and a lot like the interesting , challenging parts of programming.   
  
Like many other people I know in math grad school I really disliked all the math classes I took before college except geometry which had some proofs. There is nothing as unpleasant and unenlightening than learning rote rules to solve equations, calculate integrals or do trigonometry. I was so annoyed by the lack of compelling explanation in my calculus class that I went out and bought a real math book that actually explained things and eventually this lead me to give up physics and switch to math.  
  
Of course starting as a grad student I had very strong ideas about teaching as many people who enter math do. I wanted to do better than all the people who had taught me in college and really explain what was going on to my classes. I still try to do some of this, and take every opportunity I can to explain things when a student who wants to understand asks me, but I learned that this approach just doesn't work for most students.  
  
To put it in terms that should make sense to a programmer it comes down to the difference between programmers and computer haters. If you wanted to explain how to work some program on a computer or how to change some configuration setting you need to use completely different tactics when talking to a programmer and a computer hater. You can tell the programmer the idea behind what he is supposed to do and he has the knowledge and more importantly the desire to figure things out that will let him fill in the details. If your mother who hates computers asks the same question you need to give her a rote sequence of steps to apply giving each one in excruciating detail. Trying to make students really learn math rather than just apply rote algorithms is about as effective as taking a class of computer phobes who want to learn how to use a computer and demanding they learn what is going on the way a techie would and don't just use rote lists of steps.  
  
**It doesn't what exercises you tell someone to do or what explanations you give if someone isn't interested in understanding they won't ever learn to do more than the problems you give them**. Really learning math is about fitting together the pieces to see why things really work. If you just want to get your homework done and don't care why things work you don't even have the right pieces in your mind for an explanation to fit into. Just as the computer industry has given up on making the user really understand what is going on and instead tries to design the interface to accommodate someone who doesn't want to learn the mathematical education system has had to do the same. Some math professors still try and teach their calculus students what is really going on by making them do proofs and otherwise really require understanding but every time it fails horribly. The students give up because they don't see any connection between what the professor is saying and what they view as the point of the course, solving the problems, and even when they realize they need to learn something more to solve these problems they simply can't figure out what is important or how it fits together and just flounder horribly. Worse since the students aren't really interested in understanding the facts they are told don't fit into any mental model meaning they just can't remember all of it. Also confidence is essential to math, you need to grope around in the dark with potential solutions until one works. The biggest trouble students have is being convinced they can't do the problem and not trying but if you try and make the students do real math with real thought they rebel and just give up on them (either they are so simple they come up with a rote algorithm that almost always works and you haven't gotten anywhere or they are  
are sufficiently hard the students just flounder).  
  
You might think you should just start earlier and the problem is that by college the students have already internalized the rote approach to math. I think there may be something to this but we tried starting with understanding very early and it failed utterly. This was what new math was and the curriculum was actually very good but teachers didn't really understand it (they had learned by rote) so they couldn't implement it and parents and teachers demanded rote math back. More broadly it's not clear that everyone is really able to understand math. Different people have different abilities and understanding even the most basic math is actually pretty tough for most people.  
  
Anyway the upshot of this is that you just \*can't\* teach math the way math and programmer types would like as it would just end up failing everyone else. We have to teach rote math the same way we have to teach people rote procedures to use word and excel because they need to use the tools. I strongly support a tracking system for this sort of thing but unfortunately before college the teachers don't understand math well enough to teach it in any way but the rote way.  
  
I agree that much of the utility math we teach is totally useless. I mean no one really needs all that practice integrating weird trig functions, either you need to really understand it and you can then just figure it out or you can use mathematica. Hell a lot of college students don't need calculus and graphing functions is way overrated. However, these courses are required for the simple reason that passing them shows a certain level of ability and dedication and since they keep mathematicians employed who am I too complain. I really would prefer if they would give the two subjects different names so people didn't think I just integrated really hard functions and I want strongly tracked courses in college . But apart from that I'm not sure what can be done.

[5:42 PM, DECEMBER 02, 2006](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#5797855197940106035)

Blogger [Sarajevo06](https://www.blogger.com/profile/01550463038573419919) said...

Math is not easy out of simple reason that it has mostly been created by people that can be called geniouses.   
  
1. To even understand advanced math one needs higher IQ than most of mankind has and it is ridiculous that educational system allows that math is being thaught to so many people, while most of them neither have mind nor will to study it. Higher IQ means 110 and above which only about 40 % people have. Higher IQ - less time you need to catch up. Most people need eternity (10 t0 20 years of constant learning) to REALLY learn math. It is possible - but does it pay off for most of us ?   
  
2. In order to learn math + all other subjects in school average human must sacrifice youth from age 11 to it and give up many things most of us consider important. Life-style must change to suit everday math excercises, along with perfect mental and physical health, proper diet, lots of sports and minimum social life to avoid becoming psycho. In communist countries that was organised by state and such youth was considered national heros along with sportsman, chess-players, and military/space engineers. In the west one can easily slide into poverty, end up with miserable wife if at all with any, and be low life marginal player in society. Most of mathematicians ended up like that in history. Few of us are ready to do that. It's better and easier to start up some business and learn lots of money and get some decent sex life than be math experts.  
  
3. Math learning starts with arithmetics, and for most part after that one must learn loads of basic terms and facts before you actually start to understand anything. My method was to learn basic meaning of every term, axiom or function in my own everyday language, to expalain it descriptively, before going to formal proofs and theories. And than one must do zyllions of problems to deepen the knowledge and like that for 20 years. Good luck - average loosers, and congratulations geniouses.

[12:12 AM, JANUARY 14, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#6758176469844581769)

Blogger [Ndaru](https://www.blogger.com/profile/11716367100483892861) said...

I'm lauzy at math, thus suffer from lack of good algorithm and optimization. I guess I depend too much on existing routine libraries.  
  
Anyway, there is a good place for programmer's math:  
  
[full-text mathematics books](http://www.freetechbooks.com/forum-38.html)  
  
and operations research, if you're into queueing theory:  
  
[full-text operations research books](http://www.freetechbooks.com/forum-54.html)

[8:09 PM, JANUARY 16, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#6259630950233346759)

Blogger [paulo](https://www.blogger.com/profile/10688975557604624984) said...

thanks for the tips i ways quite in disarray trying to sort what should i improve on first.   
  
i was just wondering if any of you guys can help me find a site that teaches how read math notation.

[3:32 PM, JANUARY 29, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#4197971210668517022)

Blogger [Sunita](https://www.blogger.com/profile/17650113392440039636) said...

Logic Book:  
See Logics for Computer Science  
by A.Singh, publioshed by PHI

[10:04 PM, FEBRUARY 27, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#7791289735193813851)

Blogger [Jason](https://www.blogger.com/profile/09907406685059572339) said...

you are simply brilliant!!

[5:28 AM, MARCH 06, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#8615603540041319472)

Blogger [gorauskas](https://www.blogger.com/profile/05805866365928749435) said...

Great presentation on the design of programming languages and how math plays a crucial role in it: <http://carcaddar.blogspot.com/2007/03/complete-computing-system-in-20000.html>

[10:45 PM, APRIL 02, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#2801850687496035857)

Blogger [Renaissance](https://www.blogger.com/profile/10866530772307754358) said...

I completely, 100% disagree. I took 4 years in undergrad and more in grad school. I do programming today.  
  
Do I still solve equations or do proofs like in class? Pretty much no.   
  
Do I use the my understanding of the math I learned to solve problems regularly? Absolutely - almost daily.  
  
Do I use the math I learned to do back-of-the-envelope calculations? Daily!  
  
Could I have ever have understood or been able to do those back-of-the-envelope calculations without lots of math? Unlikely.  
  
Do find I need to learn more math beyond what I learned? Yes - optimization programming problems meant I needed more. Getting into functional programming I needed more. Etc. Etc.  
  
Could the math I learned have been taught better or differently? Yes, with very few exceptions, from grade school to grad school. It's why I'm for home schooling and deeply believe that the US public school system is fatally flawed despite being from a family of public school teachers.  
  
Do I believe there are "tiers" of programmers, ranging from "vocational technicians" to "wizards"? Yes - the "no math" claim can only apply to the former class and really no one above it.  
  
Was I lucky I did learn all math I did? Yes. It may be that it pushed over the hump past diminishing returns into compounding returns territory. I feel sorry for those who never get that far. I guess you'll never know what it's like - though I'd compare it to how Neo sees the world at the end of the first Matrix movie.

[8:37 AM, APRIL 22, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#4146866516220901166)

Blogger [Mark](https://www.blogger.com/profile/09652288045145591799) said...

*"To put this in perspective, think about long division. Raise your hand if you can do long division on paper, right now. Hands? Anyone? I didn't think so."*  
  
Yikes. You live in America, don't you?

[10:49 AM, APRIL 22, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#7845934746823717793)

Blogger [Ted](https://www.blogger.com/profile/04521780412113074636) said...

I'd suggest  
  
An Introduction to Mathematical Reasoning: Numbers, Sets and Functions (Paperback)   
by Peter J. Eccles (Author)   
ISBN-10: 0521597188   
  
It's a good book and pretty cheap by math book standards. Don't expect the material to be immediately usable in programming (although it might). The broad outline of the book is:  
  
1) Mathematical Statement and Proof  
Basic techniques and induction.  
2) Sets and Functions  
Basic Set theory. After this go read about relational database theory.  
3) Numbers and Counting  
If I remember right, this is where uncountability of reals is proven. After this, might I recommend, "What Every Computer Scientist Should Know About Floating-Point Arithmetic" (http://docs.sun.com/source/806-3568/ncg\_goldberg.html)  
4) Arithmetic  
Try some polynomial long division after this.  
5) Modular Arithmetic  
Go implement Diffie-Helman key exchange. Look out for the bit about certain numbers needing to be prime. If you do that, you might become interested in group theory.  
6) Prime Numbers  
Lots of fun.

[10:52 AM, APRIL 22, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#8462170930677234281)

Blogger [Tyler](https://www.blogger.com/profile/04039283718498649638) said...

Thanks a lot for this post. It just happens that I'm starting at a new company next week, and I'm going to be coding their recommendation system. (If you like x product, you'll probably like y.) Well, I coded something quick while I had some time to kill, but it was kludgy and slow.  
  
So, I did a bit of research and found that linear algebra is involved with doing it "right". So, I've been debating on the best way to learn what I need to know, and whether or not I even want to bother.  
  
So, thanks.

[11:33 AM, APRIL 22, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#9012905786086561354)

Blogger [Justin](https://www.blogger.com/profile/02206453686715958480) said...

Thank god you mentioned Kolmogorov complexity and information theory. These get left out of way too many discussions on anything. Here is an awesome lecture from Greg Chaitin that might get someone excited about   
metamath type stuff  
  
http://www.youtube.com/watch?v=Rr0fOGeS7DE

[3:02 PM, APRIL 22, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#6686344257069462622)

Blogger [Bart](https://www.blogger.com/profile/10490373418825679148) said...

If you want to learn mathematical logic and relearn your discrete math, while simultaneously understanding some of why you care, I'd recommend looking at computer science "formal methods".  
  
In particular, I've found   
John Jacky's   
[*The Way of Z*](http://www.amazon.com/Way-Practical-Programming-Formal-Methods/dp/0521559766) to be both highly readable and highly practical. (The title is a pun, BTW: the Z notation is British and thus pronounced "zed".) There are a number of Z tools out there now, and it really is just first order logic on sets (don't tell).

[3:35 PM, APRIL 22, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#150762607003972686)

Blogger [Marc](https://www.blogger.com/profile/02141565712549660658) said...

Re: long division  
Do you know that you can apply long division to polynomials? The textbook method works for (x^6 – 1) / (x^-1) = x^4 + x^2 + 1. Isn’t that cool? Division is deeper than “repeated subtraction”.  
  
Re: your choice of subjects  
There is no Best Subject List. The fields you find interesting are heavily influenced by the kind of problems that you are & have been working on. Ask a programmer working in mathematical finance, or on weather modelling, if calculus is useful or not.  
  
Of course, your point wasn’t to draw the draw the Perfect List, but to explain how to approach the subject. And I totally agree with you. Math should be useful and fun, and its origins in “the real world” should be made clear.  
  
Some math will be relevant for what you do; it will seem easy to learn, lively and useful. If it is irrelevant you will have difficulties creating a mental model of what is happening, and it will just seem to be mental masturbation.

[6:51 PM, APRIL 22, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#4547757738080876413)

Blogger [Kalid](https://www.blogger.com/profile/04195796758857502052) said...

There's a similar post here on the difficulties of teaching math in an overly formal way:  
  
http://technocrat.net/d/2007/4/3/17225  
  
Teaching math (or the difficulties therein) has been a passion of mine for a while. Here are a few things I've come to realize:  
  
1) Enthusiasm is huge. Having fun is important. It's hard to get people excited about a subject if you aren't. When I think about the gradient of a vector function, I think about the easiest way to cook the pillsbury dough boy. You need something better than "maximize the value of this abstract function".  
  
2) Why don't people like math? It's cumulative. If you miss one lesson (say algebra), then geometry becomes hard. Then trig becomes hard. Then calculus becomes hard. Most subjects like history, English or even Science (bio, chem, physics) are somewhat orthogonal, where you can learn one subject without the others.  
  
3) We need to know "why", not just "how". There are so many ways to understand a concept like the pythagorean theorem. I suffered through many teachers who taught rote, overly-formal proofs without giving an idea for the big picture.  
  
A real-life example: I gave a speech at toastmasters last week on adding the numbers 1-100, of all things. I used a math explanation (pairing numbers) and physical one (making a rectangle out of two triangles).  
  
It was risky, as math isn't a subject that people like, but was actually well-received! I spoke about the Gauss story, and there's a corresponding blog post:  
  
http://betterexplained.com/articles/techniques-for-adding-the-numbers-1-to-100/  
  
The key is that there's many ways to understand a concept, and getting an intuitive feel for a subject is hugely important.  
  
Just my $.02.

[7:35 PM, APRIL 22, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#2027032203092596411)

Blogger [fauxscot](https://www.blogger.com/profile/10974230958806112379) said...

tniykqI'm with you, Stevey. Learning math is a lifelong pursuit, and you can start any time.  
  
My personal favorite collection of math papers, which is useful for getting the 'breadth' of which you speak, is "World of Mathematics", but James R. Newmann. The 1988 Microsoft Press reprint version is the best one, although the originals can be found more easily.  
  
The series covers 4500 years of math, and is 133 paper from the original mathematicians. (It's really fun to read something by Archimedes (The Sand Reckoner) or Galileo.) Newmann's commentary is the math equivalent of Alistair Cooke's Masterpiece Theatre intros, and puts the paper in perspective.) Four volumes takes about 2 years of gentle reading to complete, but is worth it for the casual student. Caveat: Originally published in 1954 or so, it misses the last 50 years, but does do a good job with the first 4500!)  
  
Also, I have to expound my Hypothesis of Equal Ignorance: Given the size of the body of knowledge, we are all essentially equally ignorant. The most learned among us has only the tiniest smidgen of content.   
  
The path to increased knowledge is constant study, and any approach that meets your needs is valid.

[3:49 AM, APRIL 23, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#7564726498674871981)

Blogger [D](https://www.blogger.com/profile/06318842668023250204) said...

Two useful books on Mathematical logic, which can largely be read concurrently, are:  
  
1) Introduction to Mathematical Logic by Elliott Mendelson  
2) Computability - Computable Functions, Logic, and the Foundations of Mathematics by Richard L. Epstein and Walter A. Carnielli  
  
The former is a good textbook on mathematical logic, providing a solid introduction to the subject. The second is a great introduction to the links between mathematical logic and computability and includes plenty of content on key computational theory concepts like Church's Thesis and Turing machines.

[4:42 AM, APRIL 26, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#6238132307453532543)

Blogger [smr](https://www.blogger.com/profile/17362842977953300002) said...

Very nice article. Talks about all the right things.. And most of it is deja vu for me, like when you talk about the summation and forloop. I learned to understand summation as for loop, the hard way.  
  
I have stumbled on other posts of your blogs also a few times. You have lots of good thing to share. Its sad that you are not writing anymore.  
Do consider writing again.  
  
- [smr](http://smr.co.in/)

[6:33 AM, APRIL 28, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#7896813395459843326)

Blogger [Animesh Sharma](https://www.blogger.com/profile/07485653184282867837) said...

Thanks for this nice blog, I also get into the recursive wikipedia dive when I get stuck on some problem. On the subject of blog, my favorite is "Probability and Statistics for Computer Science"   
[http://www.amazon.com/Probability-Statistics-Computer-Science-Johnson/dp/0471326720] by James L. Johnson.  
  
Animesh  
http://computationalbiologynews.blogspot.com/

[11:47 AM, JUNE 10, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#7096606650881686616)

Blogger [David](https://www.blogger.com/profile/17892896090092693433) said...

Steve - great post. I'm also in the same situation as you were when you started, wanting to get deeper into the research papers but glossing over when hitting the harder math parts.  
  
This is the funniest (and insightful) comment I've read about math in a long time:  
  
One great thing that soon falls by the wayside is notation. Mathematical notation is the biggest turn-off to outsiders. Even if you're familiar with summations, integrals, polynomials, exponents, etc., if you see a thick nest of them your inclination is probably to skip right over that sucker as one atomic operation.  
  
  
I find myself doing that all the time. Maybe after taking my discrete math book out for a spin I can change that behavior.

[9:41 PM, JUNE 27, 2007](http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html?#1920388060544181657)

[<< Home](http://steve-yegge.blogspot.com/)

**About Me**

[My Photo](https://www.blogger.com/profile/14812997485690838920)

[**STEVE YEGGE**](https://www.blogger.com/profile/14812997485690838920)

**KIRKLAND, WASHINGTON, UNITED STATES**

[VIEW MY COMPLETE PROFILE](https://www.blogger.com/profile/14812997485690838920)

**Previous Posts**

* [Blog Or Get Off The Pot](http://steve-yegge.blogspot.com/2006/03/blog-or-get-off-pot.html)

[Powered by Blogger](http://www.blogger.com/)