

Math you need for computer science: from the Internet

- The mathematics you would like to study totally depends on the type of program you would like to work on here are a few general cases —
 1. Database Management Systems (softwares related to database management). For this one should have some basic knowledge on sets and other properties related to sets. The theory of Database management extensively deals with sets for efficient management of data.
 2. Encryption, Decryption and security related programs. For this one needs to have some basic go through with number theory and its tools like modular arithmetic. Then one also needs to learn some of its theorems like Chinese Remainder Theorem, Fermat's little Theorem, Wilson and Lucas's theorem and the like. Also if one can learn group theory it would be great.
 3. Low Level Graphics (Graphics program intended towards end users say like creating a GUI interface for a application. Note that the low level does not indicate any grade here.) For this, one need not worry much with mathematics. Maybe a bit of algebra and trigonometry.
 4. Core Graphics (Games Graphics and the likes) For this, one needs to have some good knowledge on matrices, vectors and quaternions and its properties.
 5. Dynamic Graphics (Graphics which are intended for dynamic purposes like say a graphic program which draws a 3d-Map of a street and monitors the motion of say a particular vehicle. Note that these can't be written on any High level programming language.) For this, one needs to have some intro with calculus and some interaction with partial derivatives and discrete FFT's.
 6. Artificial Intelligence For this, you need to learn graph theory which is a part of Discrete maths.
(<https://nrich.maths.org/discus/messages/2069/6386.html?1057677366>)

- 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.

...

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.

...

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. ... It's obviously important stuff, though.

- * Information Theory and Kolmogorov Complexity. 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.

...

The Right Way To Learn Math

... 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?

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!"

(<http://steve-yegge.blogspot.com/2006/03/math-for-programmers.html>)

- Mathematics is applicable in various traditional fields of engineering: mechanical and electrical engineering are among them. Mathematics is used in computer engineering too.

Mathematical logic is used in the decision making, so it is used in computer programming. As Venn diagrams are helpful in understanding the concepts of logic, they are also helpful in the programming. For instance, De Morgan's laws are used in writing statements involving decisions and Venn diagrams are helpful in understanding these laws.

Calculations are also important in the science of computers. The text you read on the computer screen is presented in a particular format. Calculations are certainly needed for these.

Geometry is used in the development of graphics. Actually a graphics screen resembles the co-ordinate plane. Just as we have points in the co-ordinate plane, we have pixels on the graphics screen. Though there are endlessly many points in any bounded part of the plane, while the number of pixels on the graphics screen is limited, yet the techniques of coordinate geometry can be used in drawing various figures on the graphics screen.

Various transformations play a part in the development of software. Two such transformations are famous as 'pop and push transformations'. As the graphs are useful in understanding different kinds of transformations, these help understand, in particular the Pop and Push transformations too.

The classical computer programming language namely 'the C language' makes a lot of use of mathematics. Different graphics commands of this language are based on the mathematical logic. The commands for making the background make use of hexadecimal numbers.

To know about the role of mathematics in computer programming in detail and to see illustrations of Venn diagrams and graphs, you can visit Mathe-

mathematics in Computer Programming (<http://ezinearticles.com/?Mathematics-in-Computer-Programming&id=2371951>)

(http://www.cameo4all.com/com_eng.htm)

- To answer your question as it was posed I would have to say, "No, mathematics is not necessary for programming?" However, as other people have suggested in this thread, I believe there is a correlation between understanding mathematics and being able to "think algorithmically". That is, to be able to think abstractly about quantity, processes, relationships and proof.

I started programming when I was about 9 years old and it would be a stretch to say I had learnt much mathematics by that stage. However, with a bit of effort I was able to understand variables, for loops, goto statements (forgive me, I was Vic 20 BASIC and I hadn't read any Dijkstra yet) and basic co-ordinate geometry to put graphics on the screen.

I eventually went on to complete an honours degree in Pure Mathematics with a minor in Computer Science. Although I focused mainly on analysis, I also studied quite a bit of discrete maths, number theory, logic and computability theory. Apart from being able to apply a few ideas from statistics, probability theory, vector analysis and linear algebra to programming, there was little maths I studied that was directly applicable to my programming during my undergraduate degree and the commercial and research programming I did afterwards.

However, I strongly believe the formal methods of thinking that mathematics demands careful reasoning, searching for counter-examples, building axiomatic foundations, spotting connections between concepts has been a tremendous help when I have tackled large and complex programming projects.

Consider the way athletes train for their sport. For example, footballers no doubt spend much of their training time on basic football skills. However, to improve their general fitness they might also spend time at the gym on bicycle or rowing machines, doing weights, etc.

Studying mathematics can be likened to weight-training or cross-training to improve your mental strength and stamina for programming. It is absolutely essential that you practice your basic programming skills but studying mathematics is an incredible mental work-out that improves your core analytic ability.

(<http://stackoverflow.com/questions/157354/is-mathematics-necessary-for-programming>)