<https://docs.google.com/document/d/1FbCWcnnajHWk594dKmN35b_we50WZf_-cwxqg-cYaRY/edit>

**Notes to (NUS) Computer Science Freshmen, From The Future**

**Intro**

Early into the AY12/13 academic year, Prof Tay Yong Chiang organized a supper for Computer Science freshmen at Tembusu College. The bunch of seniors who were gathered there put together a document for NUS computing freshmen. This is that document.

Feel free to edit it, add to it, and share it with other freshmen you know.

**Links**

You’ll probably want to start with some general advice written for undergraduates/beginners in Computing. Read:

* <http://www.paulgraham.com/college.html>
* <http://norvig.com/21-days.html>
* <http://www.catb.org/~esr/faqs/hacker-howto.html>
* <http://www.joelonsoftware.com/articles/fog0000000043.html>
* <http://www.paulgraham.com/hs.html> (What You Wish You’d Known)
* <http://al3x.net/2013/05/23/letter-to-a-young-programmer.html>
* <https://medium.com/this-happened-to-me/8f381aa6bd5e>

The sites from which the above articles are available are generally good places to read from. Also, follow some Computer Science blogs and sites:

* <http://paulgraham.com/articles.html> - Paul Graham leans towards startups but his earlier essays on programming, technology and work are worth reading. Particularly influential: Python Paradox (<http://paulgraham.com/pypar.html>) and Maker’s Schedule, Manager’s Schedule (<http://www.paulgraham.com/makersschedule.html>)
* <http://www.joelonsoftware.com/> - Start by reading his Reading Lists on the right
* <http://steve-yegge.blogspot.sg/> - Mostly entertaining, but there are a few articles that are required reading. In particular, the unofficial guide to getting hired at Google - which even Googlers refer potential hires to - (<http://steve-yegge.blogspot.sg/2008/03/get-that-job-at-google.html>) and his tour of programming languages (<https://sites.google.com/site/steveyegge2/tour-de-babel>)
* <http://www.codinghorror.com/blog/> - Generally good, techy posts by one of the founders of Stack Overflow
* <http://www.scottaaronson.com/blog/> - Theoretical Computer Science
* <http://lambda-the-ultimate.org/> - Group blog that tracks innovations in Programming Languages research
* <http://news.ycombinator.com/> - Hacker News; just about everyone in computing reads it.

**(Informal) Checklists**

Use the checklist below to get a rough measure of your knowledge.

* <http://www.indiangeek.net/wp-content/uploads/Programmer%20competency%20matrix.htm>

**Introductions to various specializations**

**Graduate School**

* Applying to Ph.D. Programs in Computer Science -<http://www.cs.cmu.edu/~harchol/gradschooltalk.pdf>
* A CS Graduate School Survival Guide -<http://www.cs.unc.edu/~azuma/hitch4.html>
* Ph.D. Grind -<http://pgbovine.net/PhD-memoir-epilogue.htm>

**Programming Languages**

* Programming Languages Research -<http://tagide.com/blog/2012/03/research-in-programming-languages/>
* Rich Programmer Food (Yegge on compilers) -<http://steve-yegge.blogspot.sg/2007/06/rich-programmer-food.html>

**Game Programming**

* So you want to be a video game programmer? -<http://all-things-andy-gavin.com/2011/08/29/video-game-programmer-why/>

**Learning Unix**

If you have a Mac, start playing around with your terminal.

If you don’t, figure out how to install Virtual Box (<https://www.virtualbox.org/)> and then figure out how to install a Linux distribution on it. Recommended ones are:

* [Ubuntu](http://www.ubuntu.com/) (Most user friendly?)
* [Linux Mint](http://linuxmint.com/)
* [Arch Linux](http://www.archlinux.org/) (minimal Linux distro. Good for those who likes to poke around and install stuff on your own. Not the most newbie friendly.)
* [CentOS](https://www.centos.org/) (Most battle tested, loved by sysadmins who treasure their sleep)

Unix is **required knowledge**. As is familiarity with the shell (figure out what that means ;-) (Also, figure out what 'sudo rm -rf /' means *before trying it out*) (And :() { : | : };: )

Learn an editor - either emacs or vim. They will save you a lot of time when you’re coding for CS2105. Actually, in pretty much everything else in life, as well.

* **Vim** - to get started, run vimtutor from the terminal , read [this](http://yehudakatz.com/2010/07/29/everyone-who-tried-to-convince-me-to-use-vim-was-wrong/), [this](http://rc3.org/2012/05/12/the-grammar-of-vim/) and [this](http://yanpritzker.com/2011/12/16/learn-to-speak-vim-verbs-nouns-and-modifiers/), and then *zhng* your Vim according to [this](http://stevelosh.com/blog/2010/09/coming-home-to-vim/).
* **Emacs** - Get emacs for your platform from the [GNU site](http://www.gnu.org/software/emacs/). Or google for more specific instructions for your platform.

It’s always good to [RTFM](http://www.gnu.org/software/emacs/manual/), and to go through an [emacs lisp intro](http://www.gnu.org/software/emacs/emacs-lisp-intro/). Or get a quick head start with the [emacs starter kit](https://github.com/technomancy/emacs-starter-kit). pew, pew, pew [smoking all the other editors ;) ]

Know some regex-fu.

**Learning Stuff**

Pick a project to trick yourself into learning new technologies. Drop by Hackerspace.SG (<http://hackerspace.sg/>) Attend NUS Hackers events (the club was created, years ago, by the same people who founded Hackerspace.SG). Intern. Sign up for GSoC (<http://code.google.com/soc/>). Work for a research lab for peanuts (only recommended if you really want to learn the techniques from that lab, or you want a career in academia).

If you need something to trick yourself into tricking yourself, take CS3216/CS3217.

Read code - lots of it - there’s a whole truck-load of it available online (a good source is [this book](http://www.aosabook.org/en/index.html), and this [list](http://www.quora.com/Computer-Programming/Where-can-one-find-examples-of-good-quality-open-source-coding/answer/Toby-Thain)).

Learn and use version control. Thank us later.

* For Mercurial - there’s a great [guide](http://hginit.com/) written by Joel Spolsky.
* For Git - read [Pro Git,](http://git-scm.com/book) chapters 1-5, and skip chapter 4. Come back and read the rest after you’ve used Git for awhile.

Set up a<http://github.com> account. ‘Follow’ a few famous [programmers](http://www.quora.com/GitHub/Who-are-the-best-coders-to-follow-on-GitHub). Start a few projects and show them off there. Also, don’t forget to give back to the community. Contribute to popular open source projects by sending in pull requests on github. *Some awesome open source NUS projects include NUSMods:* [*https://github.com/ahbeng/NUSMods*](https://github.com/ahbeng/NUSMods) *MODIVLE:* [*https://github.com/ymichael/modivle*](https://github.com/ymichael/modivle) *CORS Planner:* [*https://github.com/zhuochun/cors-planner*](https://github.com/zhuochun/cors-planner)

Consider starting an open source project.

Build your profile, write your blog using [jekyll](http://jekyllrb.com/) using [Github Pages](http://pages.github.com/) for free.

You may also apply \*free\* github student account, [checkout here](https://github.com/edu), applying using your NUS account makes it easier to verify that you’re really a student. Private repos are great for collaborating on school projects.

Hack for fun. Fun and pointless is better than unicorny and non-existent.

When in doubt, sketch out your problems. Diagram them. They'll usually be easier to break down and analyse after a few doodles. If that doesn't work, take a break.

Read The (um..) Fine Manual. *[aka RTFM]*

Question *ALL* your assumptions! eg. Does 'x' really hold the value I think it does here?

Before coding a single line, question your own mental model of the problem. Do you understand the problem? Do you understand the desired outcome?

Know the rules first, then see how you can break them.

**Good/Interesting Courses in SoC**

* **CS1101S Programming Methodology** ( Functional Programming Paradigm )
* **CS3216 Software Development on Evolving Platforms** ‘the Facebook module’: Multi-disciplinary approach to building web apps. Crazy workload (you’re expected to pick up new technologies and run with them in a week). Loads of fun. Not only for programmers.
* **CS3217 Software Engineering on Modern Application Platforms** - ‘the iPad module’: CS3216’s twisted cousin. Programmers only. Highly interesting (and time-consuming) module where you will have to code a few iOS apps within a semester.
* **CS3233 Competitive Programming:** An extremely tough module, suitable for anyone thinking of challenging themselves.
* **CS2104 Programming Languages Concepts** - an excuse to learn Assembly, C, Prolog, Python, Ruby, Haskell, OCaml, Scheme and Oz in a single semester.
* **CP3108B** (Mozilla) - Independent Project module. “The Mozilla module”. Want to contribute to open source projects but don’t know where to start? Join it. ;)
* Why limit yourself to modules? You might be surprised to know that SoC is pretty generous in allowing Independent Project modules. Want to work with an industry partner on a project (that is not in the ATAP list), just go talk to your advisor.
* **CS5231 Systems security**. You rewrite IP Tables as an assignment ;)

Colloquial book titles are used where it’s famous enough to warrant so. For instance, when people in CS say ‘CLRS’, they mean ‘Introduction

You can also browse [this (old) list of high-level SoC modules](http://www.comp.nus.edu.sg/undergraduates/cs_cs_focus.html). As a general rule, the higher level specialized modules tend to be more interesting (pending good lecturers). Try and get past the level-2000s so you can get at the level 4000s and above.

**Famous Books**

These are books that are widely recognized to be seminal in computing. Worth reading (or at least putting on your to-read list).

to Algorithms’.

* [SICP](http://mitpress.mit.edu/sicp/) (an introduction to programming that’s pretty much the best out there. *Note: not many people have finished this, though*).
* [The Mythical Man Month](http://en.wikipedia.org/wiki/The_Mythical_Man-Month) (a classic in Software Engineering)
* [CLRS](http://en.wikipedia.org/wiki/Introduction_to_Algorithms) (*the* book on algorithms)
* [Modern Operating Systems](http://www.amazon.com/Modern-Operating-Systems-3rd-Edition/dp/0136006639) (*the* book on Operating Systems)
* [The Dragon Book](http://en.wikipedia.org/wiki/Compilers:_Principles,_Techniques,_and_Tools) (*the* book on compilers)
* [The Cinderella Book](http://en.wikipedia.org/wiki/Introduction_to_Automata_Theory,_Languages,_and_Computation) (*the* book on the theory of computation)
* [Russell & Norvig](http://aima.cs.berkeley.edu/) (*the* book on Artificial Intelligence)
* [Paradigms of Artificial Intelligence](http://norvig.com/paip.html) by Norvig
* [Gang Of Four](http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612) (*the* book on Design Patterns)
* [The New Turing Omnibus](http://www.amazon.com/New-Turing-Omnibus-Turning-Excursions/dp/0716782715) (a preview of *everything* in Computer Science)
* [The Art of Computer Programming](http://en.wikipedia.org/wiki/The_Art_of_Computer_Programming) (a review of *everything* in Computer Science; pretty much nobody, save Knuth, has finished reading this)
* [The C Book](http://en.wikipedia.org/wiki/The_C_Programming_Language) (worth reading for its style, influence and brevity).
* [The Pragmatic Programmer](http://pragprog.com/the-pragmatic-programmer/)
* [The Passionate Programmer](http://pragprog.com/book/cfcar2/the-passionate-programmer) (My Job Went to India)
* [Pragmatic Thinking and Learning](http://pragprog.com/book/ahptl/pragmatic-thinking-and-learning)
* [Rework](http://www.amazon.com/Rework-Jason-Fried/dp/0307463745)
* [Hackers & Painters](http://www.amazon.com/Hackers-Painters-Big-Ideas-Computer/dp/0596006624) by Paul Graham
* [Clean Coder](http://www.amazon.com/Clean-Coder-Conduct-Professional-Programmers/dp/0137081073) by Uncle Bob
* [Little Book of Semaphores](http://greenteapress.com/semaphores/) (Good for OS and parallel programming modules)

List of free programming ebooks -

[https://github.com/vhf/free-programming-books/blob/master/free-programming-books.md](https://github.com/vhf/free-programming-books/blob/master/free-programming-books.md#scheme)

<http://stackoverflow.com/questions/194812/list-of-freely-available-programming-books>

Contains links to freely available ebooks online. Good for those living on a FnM (Father and Mother) scholarship.

**Good books**

Not necessarily about programming/computing.

* [Godel, Escher, Bach](http://en.wikipedia.org/wiki/G%C3%B6del,_Escher,_Bach)
* [Delivering Happiness](http://www.amazon.com/Delivering-Happiness-Profits-Passion-Purpose/dp/0446563048)

**Good talks**

Some recently famous; others highly influential.

* [Growing a Language,](http://www.youtube.com/watch?v=_ahvzDzKdB0) Guy Steele (wait till 8:43 minutes before you understand what he’s doing).
* [What We Actually Know About Software Development, and Why We Believe It’s True](http://vimeo.com/9270320) Greg Wilson (*how* to think about software engineering)
* [Stats that shape your world-view](http://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen.html), Hans Rosling (on data-visualization)
* [Inventing from Principle](http://vimeo.com/36579366), Bret Victor (the future of computing interfaces)
* [The Last Lecture](http://www.youtube.com/watch?v=ji5_MqicxSo) - Randy Pausch (Achieving Your Childhood Dreams)
* [2005 Stanford Commencement Speech by Steve Jobs](http://www.youtube.com/watch?v=D1R-jKKp3NA)
* [Harvard Commencement Speech](http://www.youtube.com/watch?v=wHGqp8lz36c) by JK Rowling

**Seriously Good CS Lectures:**

* [CS Lectures from Stanford](http://see.stanford.edu/see/courses.aspx), and especially:
  + [Programming Methodology](http://see.stanford.edu/see/courseinfo.aspx?coll=824a47e1-135f-4508-a5aa-866adcae1111)
  + [Programming Abstractions`](http://see.stanford.edu/see/courseinfo.aspx?coll=11f4f422-5670-4b4c-889c-008262e09e4e)
  + [Programming Pa](http://see.stanford.edu/see/courses.aspx)[radigms](http://see.stanford.edu/see/courseinfo.aspx?coll=2d712634-2bf1-4b55-9a3a-ca9d470755ee)
  + These lectures will would really push you. And if you are up for it do the assignments too.

Programming Languages Lectures

* Douglas Crockford - [Crockford on Javascript](http://yuiblog.com/crockford/)
* Rich Hickey - [Series on Clojure Design and Thinking about Programming Languages](http://www.infoq.com/author/Rich-Hickey;jsessionid=FDE428234E8B76D971CA0984D7A0A194)

**Advice from seniors**

*Seniors - fill this in please*

Adhiraj - Computer Science is not about programming, though they go hand in hand.

Ced - Don’t be afraid.<http://metacog.tumblr.com/post/15889275145/fear-of-programming-stupidity>

Div - You’re probably not studying as much as you should be

Wen Bin - join CCAs, don’t just mug. CAP is not everything.

Vaarnan - Be language agnostic. Learn how to self-learn. Also, the curriculum is made for everyone to follow. Learn beyond the course.

Frank - Experiment; it’s not expensive to experiment with computers.

Laurence - Go out and meet people from industry

Jason - Be brave. Read, read and read. Read widely. Don[’t just read about computer science stuff.](http://www.ted.com/talks/derek_sivers_how_to_start_a_movement.html) ([Derek Sivers](http://www.ted.com/talks/derek_sivers_how_to_start_a_movement.html), has a very good reading [list](http://sivers.org/book)) Learn at least a new language every year (Or learn [7 Languages in 7 Weeks](http://pragprog.com/book/btlang/seven-languages-in-seven-weeks)). Make sure the language messes with the way you

normally think about programming. [Alan Perlis](http://pu.inf.uni-tuebingen.de/users/klaeren/epigrams.html) once said: "A *language* that doesn't affect the way you *think* about *programming*, is not *worth* knowing"

Andrew - Make stuff to make your life easier. This works in more ways than one.

Silin - don't be intimidated/overwhelmed if you find yourself among the really awesome programmers, especially if you're coming in without any background. do your best to keep up with the lectures, and learn to pick up skills/knowledge outside of school-taught content by doing your own reading

Yiping -

1. Learn it the hard way. CS people are not meant to write fancy web pages to impress people from business school. Get yourself deep into the technical stuff. A sound foundation in C programming and system architecture are essential.
2. The teaching style in NUS (at least undergraduate courses) is easy to follow but often omits some exciting challenging stuff. Get yourself used to reading the textbooks (not the garbage ones) from cover to cover and to read research papers.
3. If you want to get an admiring CAP, the best way is to **forget about your CAP**. Choose the courses that appeals to your heart and can help you in your research.

Jesmond - Don’t be afraid to challenge yourself. Some of you may be saying to yourself “I have no background” or “The workload of the module seems daunting” but in the end you’re only preventing yourself realising your potential (just to let you know, I had never written a single line of code prior to entering uni and I’m surviving fine). Only through the ‘hardcore’ modules do you learn skills that are practical for real-world use. Have a little confidence in yourself. University is the last place you can make mistakes without reservation before you go out into the working world. Don’t learn what you need to know to pass exams because very quickly that knowledge may become obsolete. Learn how to learn. In computing, it is very hard for you to be able to dig a niche for yourself and get away with it. Technology evolves at a mind-blowing pace and the only way you can keep abreast is if you’re able to evolve with it. Pursue what piques your interest because that’s the only way you’ll be able to keep yourself going onc e the going gets tough. Learn to read the f\*\*\*ing documentation. It’s the best way to learn the nuances of a language. And last but not least, Google is your best friend =D

Shubham -

1. Ask questions. Don’t be afraid. At max, the other person will refuse to answer and never talk to you again. If the latter scenario happens, it is probably as well since he wasn’t helping you much anyway.
2. Learn to say ‘no’. I have learnt it the very hard way that doing okay in many things and badly in a few is exponentially worse that doing well in one and only one. Don’t be overwhelmed by the many things people around you are doing. Believe and have faith in yourself.
   1. Stay fit. You shouldn’t ma
   2. Richard - CS can get very political, and it's easy to get into debates where someone says you should make use of some software/programming-language/API/operating-system/convention etc. etc. rather than the one you're using, or how you should learn 20 different things when what you know is all you need to get you throughke programming late at night an excuse for a Mc Donald’s meal or a coke at night. Do at least half an hour of vigorous exercise everyday. Never think you don’t have enough time to care for your body. A healthy body also allows your brain to work faster. Again, saying this from personal experience.

the job. (And they may even be correct. ;-) ).

Ostentation will not make you a better programmer.

Ryan - Don’t just spend all your time on school work. CS is all about doing things yourself, figuring things out without much official help. Hack around whatever thing you like. Remember, the goal is learning. You can’t learn with just textbooks alone. You must learn by doing. Also, make yourself surrounded by smart people. You can learn a lot from your peers who are better than you.

GCL (aka Chun Lin/Carrot) - Read newspaper or books (not just textbooks), everyday.

Bach - Learn different kinds of programming paradigms. Learn functional programming, logic programming, stack-based programming, voodoo-oriented programming (I made the last one up)... The (computing) world is not just made of this shiny OOP thing.

Joshua - It doesn’t matter what you study, as long as you keep learning. Read lots (blogs, books, papers), keep experimenting and keep practicing!

Eldric -

1. Differentiate yourself from the rest. Don’t make yourself a cog in a machine. If you do, make yourself an important one. If not you will be replaced with someone cheaper. Hint: do your own stuff outside of class
2. Study other disciplines. Computer science is a great aid to other disciplines ⇔ you understand enough of the problem in computing terms to solve it.
3. Pick up logic - If nothing else, **make sure you don’t forget the natural deduction and first order logic you learn in CS1231 (to freshies: make sure you LEARN it.)**. Most papers are written by people who feel like throwing in excessive symbols to represent simple things.
4. Make your school fees worth it. I.e. don’t spend time taking easy mods to pull your CAP up. I should repeat that this is basically the last time that you
   1. get to make mistakes and get away with them
   2. get to learn (additional) things for for free.
5. Study ideas, apply technologies. thing on thThe hottest e market now a) is going to continue being the hottest thing, b) get chucked out.  
   If a), then well everyone will know it, violating 1). If b) that means everyone that knew it found something better, so no loss either. Just learn enough about a technology to apply(unless specializing into tech field, e.g. security). FYI, cloud computing is almost as old as the internet ;)
6. That which does not exist is for you to create!

Shaun Stanislaus -

Ideas were never really new, they were reinvented.

Yang Shun -

1. Learning how to learn is crucial in this age where current technology turns obsolete in the blink of an eye. The skills you possess may not be needed by your future employer and you will have to learn on the job. Your employers will want you to learn FAST.
2. Like what you do and do what you like. Naturally you will spend more time on it and you will do them well.
3. CAP is hardly of importance (but don’t neglect it!) in securing a decent job. Your employers look out for your past experiences and how fast you pick things up. Attitude comes before aptitude.

Viet -

1. Computer Science/Software Engineering is the best industry where you can fail early, fail often, and fail cheaply, and learn new tricks more efficiently
2. Inspiring video by Derek Sivers: Why you need to fail<http://www.youtube.com/watch?v=HhxcFGuKOys>
3. The test of pudding is in the eating. Go get a freelancing project, teach younglings and be IT consultant. You will learn more from what you picked up from theory books and grill your knowledge and skills
4. The ultimate goal of science and engineering, including Computer Science & Software Engineering is to reproduce success and best practices so learn to convey your message with simplicity

Minqi -<http://ymichael.github.io/projectintern> on overseas internship. Also, it has been mentioned before but again, learn and *use* a version control system. I have seen people spending 1 year as a computing student, yet still refusing to learn git, using Dropbox to sync code with their teammates...