Essentials of Application Software Development

# Introduction

This document details a self-paced learning course, which will help you become a well rounded application software professional. A professional coveted by employers, managers, customers, co-workers, everyone.

## Course Overview

This course is designed to help people with minimal background of computer software to quickly pick up skills of the software trade. They need to do this primarily through self-effort, and expect to transform into a professional with whom even the best out there would love to work. It covers core technical areas, industry specific aspects, as well as soft skills and life skills. And all these are covered at a level of depth most appropriate to what we term as: *‘a generic software professional with mastery in some areas, and who can efficiently self-train’.*

It is not an exhaustive degree course in Computer Science, but kind of a mini-course covering areas that matter most with respect to software industry needs observed most commonly. Here one gets productive quickly without spending the time on going through a full-fledged degree course. The other stress is on imbibing capability and confidence in learners to hack their way and learn up any area of software as and when needed. In this internet age ample study material is available at the click of a button. So, continuous incremental knowledge buildup through part-time study via online micro-courses or full-on self-effort, is feasible.

## Who Could Benefit, How?

* Aspiring software professional: Those with or without formal educational background in software, not working yet, but aspiring a rewarding career in this field. Either through a software job, freelancing engagements, or other forms of self-employment.
* Entry level software professional: Those who have just started their software career, and wanting to expedite their career growth in the initial phases.
* Working Software professional: Who have been working in software for some time, but feel it is not working too well for them probably because they didn’t start too well. They would like to take a step back and revisit a few things.
* Student, or people from a non-software stream: Understand how software works at sufficient breadth and depth. Learn many useful things to effectively leverage software in your area of work. Either by yourself, or through collaboration with other software teams.

## Prerequisites

* Basic to intermediate level understanding of English.
* Basic awareness and ability to use computers or smartphones.
* Understanding of basic high school level mathematics.
* Reasonable maturity to self-study with some help. Typically over 14 years of age.
* Access to a Windows/Mac/Linux desktop/laptop, and internet.

## Course Details

* This is a comprehensive course covering all important aspects of Application Software Development. System level concepts are also covered at a high level, to the extent that they are needed for application software development.
* Self-paced course which you should be able to completely mostly through self learning, and a bit of external help.
* Balance of theory and practice, along with training on professional execution of real-life projects.
* Python is the mainstream programming language used throughout the course. Other languages such as Java or C are covered to the extent necessary, such as understanding code snippets in java, or level code in C/C++.
* The core philosophy is that, if one learns up fundamental and a few other pivotal areas well, knows other areas at a high-level, and develops the skill to find appropriate learning material and self learn from it, anything else can be learned as required. It is not necessary to pre-learn so many things just because those might be needed sometime. In our opinion, a big waste of brain cells and time spent on learning things you are less likely to need. In today’s digital world there is absolutely no dearth of high quality learning material floating freely over the internet.
* The study material we have selected is sometimes very extensive and demanding, and sometimes at a higher level, without the details you would generally find in a full degree course. We have made this selection based on what is most fundamental and needs to be done to perfection, since one will encounter those things most frequently going ahead. As opposed to what would be more likely to be useful for specialized requirements. We have tried to minimize time spent on things that are less likely to be encountered by a typical software professional, yet without diluting important aspects based on which other skills can be picked up easily. Kind of trying to get you the best of both worlds.
* Content mix at varying level of detail
  + Must-have areas covered in sufficient detail.
  + Overview of other less pervasive areas in order to put them in context ,so that they can be learned in detail as/if needed.
  + Rigorous coverage of a few chosen complex technology and research topics, to improve confidence on comprehending tough topics.
  + Hands-on training on real-life projects and problems.
  + Core Technical areas, Industry specific practices, as well as Soft skills and Life skills.
* Duration: 365 days for those with a bare minimum background. Time will reduce depending upon academic background and sincerity with which they have done their academics. Students can skip topics which they know well already. Find come correlation to this extremely rigorous *MIT Challenge* by Scott Young: <https://www.scotthyoung.com/blog/myprojects/mit-challenge-2/>.
* Material: Documents, Videos, Exercise problems, Programming and other software related assignments, Projects.
* Self Evaluation: Comparison with answer keys, Online programming.
* Contents:
  + Learning to self-learn, Clear understanding, Clarifications, Problem solving.
  + Technical:
    - Introduction: Electronics, Computers, Hardware and Software, Information Technology
    - Basics of: Mathematics for Computer Science, Digital electronics, Digital Logic, Binary number systems
    - Computer Usage, Useful Commands, common tools.
    - Fundamentals of Programming in Python, and a bit of Java and C/C++.
    - Data Structure and Algorithms, more Programming
    - CS foundations: Maths, Automata theory, Architecture and OS.
    - Advanced programming techniques: Libraries, Parallelism, Asynchronous, Decorators.
    - Frequently needed Computing topics: Databases-SQL, Networks, OS, Software design-engineering
    - Overview of specialized topics: Graphics, AI, Functional Programming, Embedded Systems, Compilers, etc.
  + Design and Development practices:
    - Design and Development Tools
    - Project Management methodologies
    - DevOps, Continuous Integration, Monitoring
  + Focussed self-learning tracks: Specialization in chosen tracks, study of open source projects, mini projects: N-tier web apps with load balancing, Html/Css/Javascript based client apps, and more to be added.
  + Complex topics through popular research papers. E.g. Google BigTable, MapReduce, Turing Test, Large scale Search Engines, Virtualization, Single Chip Microprocessors, etc.
  + Soft skills, Life Skills:
    - Activity Organization, Communication, People skills, Presentation skills.
    - Environment touch, General awareness: Technology, Business, Social, Political, Economic.
    - Business understanding, Entrepreneurship.
    - Leisure, Investing, Relationships, Fitness, Spirit and attitude.

## Instructions to Learners

* Those who want to complete this course thoroughly should go through individual sections religiously. Go through articles, videos, programming exercises and other problems. Others can use this in whatever way they deem fit.
* In general, try to thoroughly understand what you read-see-hear. **Do not hurry** through a topic just in order to finish it. **Take it easy.** Avoid having a long backlog of things you have not clearly understood at necessary depth. Search the internet to know more around something that you are curious about, if the study material does not cover it. As Hitopadesha says: “*A wise man should pursue knowledge and wealth as if death and old age did not exist. He should practice virtue as if death had seized him by the hair of his head*”
* In case you are already comfortable with some topic, feel free to skip it. E.g. Usage of computers.
* There might be situations where you are stuck with an explanation, don’t understand something, or don’t know how to proceed further. In such cases, take help from Google, Knowledgeable people around you, or internet forums. Seek help with sincerity and boldness.
* We will do our best to keep the links in this running document up to date. But as time passes (or you are using a non-current version of this document), links provided here might break. You then need to find out equivalent articles on the web. You need to gradually develop the ability to discern effectively between articles of substance, or pure noise or hidden advertisements of a particular technology. This habit has overarching effects in other fields of life as well.
* In general, whenever you are not able to understand a particular topic from an article or video, these could be different reasons for it. Either the explanation assumes some prior understanding, the language or structure of explanation is complex, you are not in full attention. Try to figure these out for yourself and redo those parts, or seek help appropriately. There could be situations where the quality of explanation itself is sub-standard (and we have tried our best to avoid including such here), in which case, please bring it to our notice, and we will try to improve our compilation.
* Try to associate with others who are learning what you are, whichever way you can. Discuss with them. As this ancient saying goes: *The student gains one part of knowledge from association with the preceptor (in your self-learning case, professor videos, books, google, experts on internet forums are distant next-best alternatives). The second part comes through enthusiasm or effort with the awakening of a particular kind of intelligence. The third part comes just by the influence of time. (In the course of time, the student's intelligence matures). And the last part is achieved through interaction with co-learners.*
* The CS Fundamental areas and Commonly Needed Topics should be covered, before the Electives and Detailed study sections. Within those initial sections, it is recommended to go more or less in sequence. The programming and algorithms sections should be done sooner than later. Other than that, you can go through sections in whichever order you feel comfortable, depending upon your inclination and mood.
* We suggest that you spend an hour or two to read this entire document from start to end once. You will understand how various things are structured, and then start learning in detail from the beginning, and decide in which sequence you would want to go.
* It is extremely important that you attempt the exercises and programming assignments given here in all earnestness. Many problems have solutions provided. However, do not give up quickly and look up the solutions unless you have spent enough time trying out various ideas to solve the problem. It is perfectly ok if you are not able to solve, but attempting by trying out various ideas on your own is most important. Listening videos, reading articles is fine and useful. But there is a world of difference between learning from a football match and playing football (especially if playing football is going to be your regular job :) ).
* Lessons also contain exercises which depend upon how well you have grasped information. You cannot ‘solve’ such. Either you know enough to answer, or you don’t. If you don’t, revisit study material or google for the topic.
* Code examples are presented in Python and Java as far as possible. However, if equivalent Java examples are not provided or not comprehensive enough, learners should search for such on the internet, based on Python samples.
* Read a lot of code with alertness and earnestness. Small snippets also embed important and peculiar concepts.

## Special Mentoring, Certification

Learners might find parts of this course difficult to understand. We have consciously not provided over-simplified learning material just because it is easy to understand. We felt we should not compromise on rigor and detail wherever it really matters. For such situations, we offer guidance (not to be confused with spoon-feeding) to those undergoing this course, both online and personal. Guidance ranges from holding sessions for entire course contents, or only for subsets as per need and interest, or only doubt clarification sessions.

Other options like computer and internet infrastructure, explanation in local language, classroom space, on-premise course delivery, and certification for satisfactory completion of this course, are also available as paid offerings. From us, as well as by other teachers and institutes. Please contact us to know more about these.

## Feedback, Comments, Corrections

We highly appreciate sincere feedback, comments, and corrections to this document. These will help us keep improving this course on a continuous basis such that it continues remaining useful over a long time period.

Please drop us a message with your thoughts to: xxx@xxx.com.

# Preliminaries

Duration: 15 days.

## Understanding Spoken English

Many video lectures here are by native English or American speakers. Students from other geographies might encounter difficulty understanding this. Go through some of these links to improve your understanding of English, and the native accent.

* People of different ethnicities teaching how native English speakers speak, peculiarities of pronunciation. Go through one of these as per your own cultural background.
  + <https://www.youtube.com/watch?v=4tGbezSQrtY>
  + <https://www.youtube.com/watch?v=E2i3hjR_SIU>
  + <https://www.youtube.com/watch?v=nigdf69TSNE>
  + <https://www.youtube.com/watch?v=6pxUmP82aQw>
* Further exposure to how native english speakers communicate:
  + <https://www.youtube.com/watch?v=nRiHKI_8u0A>
  + <https://www.youtube.com/watch?v=RdcwmKiSQLg>

## Computers and Information Technology

This section has an overview of computers and information technology, also gets you familiar with common computer work environments (operating systems).

Watch playlist on basics of computers, with an introduction by Bill Gates: <https://www.youtube.com/playlist?list=PLzdnOPI1iJNcsRwJhvksEo1tJqjIqWbN->

Watch more videos on Computer hardware and software: <https://www.youtube.com/watch?v=8UyJMiYqvs4>, <https://www.youtube.com/watch?v=gaN1SKti3ts>

Go through appropriate tutorials on the web and get used to a couple of common operating system environments. Pause videos at various points and try out the actions shown in the video, on your own computer. Just play around.

1. Watch video on basic computer usage with the Windows Operating System: <https://www.youtube.com/watch?v=DwsKeoXOa9I>
2. Now check out Linux, another fine operating system, through this: <https://www.youtube.com/watch?v=PTaL1s3YJPY>.

Computer tricks that everyone should generally know: <https://www.techspot.com/guides/676-best-computer-tricks/>

We recommend doing this entire course in Linux OS, but if you are comfortable with windows or Mac, that is ok too.

### Test

Try out this quiz: <https://www.allthetests.com/quiz31/quiz/1418142224/Basic-Computer-Internet-Knowledge>. Use google to look up more, if the above training material did not cover some topic there.

## Self Education, Getting Help

This course is has self-learning as an important aspect.

Get to know about self learning first, through: <https://99u.adobe.com/articles/29995/never-stop-learning-how-self-education-creates-a-bullet-proof-career>, <https://www.nateliason.com/blog/self-education>, <https://www.scotthyoung.com/blog/2007/11/12/how-to-teach-yourself-anything-in-less-than-three-months/>.

As you do this course, there will be enough instances when you are stuck and need help. So it is very important to understand various lifelines available to you. Get comfortable with them first.

Learn to search the web, google: <https://www.youtube.com/watch?v=LrSXuEmN8O0>.

Get familiar with sites: stackoverflow.com for help with technical queries: <https://simonecarletti.com/blog/2016/12/how-i-use-stackoverflow/>.

Get familiar with quora.com for general questions you may have: <https://www.quora.com/How-do-I-get-started-using-Quora>.

Get familiar with reddit.com. It is similar to Quora in some sense, but also different and fun to use once you get the hang of it. <https://mashable.com/2012/06/06/reddit-for-beginners/#oeUHCv7eTEqo>

## Under The Hood, Electronics, Digital Technology

It is always interesting to intuitively understand how a certain thing works, starting from natural principles. It is not necessary to understand those things in detail in order to understand software well, but one should have an overview for how things work under the hood.

Get some basic knowledge of various electrical and electronic components, and circuits by going through: <https://www.instructables.com/id/Basic-Electronics/>

Get to know about semiconductors, on which almost all components of a computer are based on: <https://electronics.howstuffworks.com/diode.htm>

Understand the binary number system. Either see videos from <https://www.khanacademy.org/math/algebra-home/alg-intro-to-algebra/algebra-alternate-number-bases/v/number-systems-introduction>, or read here: <https://code.tutsplus.com/articles/number-systems-an-introduction-to-binary-hexadecimal-and-more--active-10848>.

Understand what bits and bytes are, their relevance to binary system and computers: <https://computer.howstuffworks.com/bytes.htm>

Understand the difference between digital and analog signals: <https://learn.sparkfun.com/tutorials/analog-vs-digital>

Understand how a chip made up of silicon and wires help achieve all that a computer does, through what is called boolean logic: <https://computer.howstuffworks.com/boolean.htm>

Understand how semiconductors are used to realize boolean logic by going through: <https://electronics.howstuffworks.com/digital-electronics.htm>

Understand what integrated circuits (ICs) are: <https://www.explainthatstuff.com/integratedcircuits.html>

### Test

Do a recap of various terms used above such as diodes, transistors, ICs, boolean logic, digital, draw an approximate mind-map depicting connections between them. This is one example: [http://ewallmaps.com/map-electronics/#prettyPhoto[gallery1]/3/](http://ewallmaps.com/map-electronics/#prettyPhoto%5Bgallery1%5D/3/).

Take the test on binary number systems from <https://www.khanacademy.org/math/algebra-home/alg-intro-to-algebra#algebra-alternate-number-bases>, or <http://faculty.cs.niu.edu/~hutchins/csci360/practice/arith-pr.htm>.

## Unix Commands and Composing Them

If you are using windows, install a unix environment such as Mingw or Cygwin. Not needed for Linux or Mac.

Learning the Unix command line is will equip you with simple tools that can be combined in many interesting ways. This will help automate common tasks that you will frequently encounter throughout your career.

Check out this old documentary on the Unix Operating System, its philosophy, from stalwarts in the field of computers: <https://www.youtube.com/watch?v=XvDZLjaCJuw>. Should get you pumped up somewhat.

Understand and practice common Linux commands through the do-it-yourself environment at: <https://linuxsurvival.com/linux-tutorial-introduction/>. Also do the quiz exercises therein. Then go through <https://ryanstutorials.net/linuxtutorial/> to improve your understanding.

Learn unix/linux shell programming/scripting here: <https://ryanstutorials.net/bash-scripting-tutorial/>

Understand regular expressions and practice commands given here: <https://ryanstutorials.net/regular-expressions-tutorial/>. Regular expressions is a very useful general purpose skill to have. Not just at the linux shell, but any kind of programming language you will use later on, of course, with some minor variations.

Useful commands: <https://opensource.com/article/17/7/20-sysadmin-commands>, SSH: <https://www.hostinger.in/tutorials/ssh-tutorial-how-does-ssh-work#gref>

Unix allows us to do sophisticated text processing using some very basic and simple text processing commands. Go through: <https://learnbyexample.gitbooks.io/linux-command-line/content/Text_Processing.html>

### Test

Attempt problems given at: <https://www.hackerrank.com/domains/shell>. Do at least two or 3 across each level of difficulty, and each subtopic learned above.

# CS: Basic Areas of Computer Science

## Fundamentals of Programming

Duration: 30 days

Do MITs CS6.001 course, that introduces Computer Science and programming using Python: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/>. Go through video lectures, in class questions, and exercises.

Also, though Python 3.x is considered to be the future, you will frequently encounter Python 2.7 code. Therefore, knowing the differences between the two is helpful. See: <http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html>.

**Optional:** Harvard’s CS50 is also an excellent course, probably more polished than MIT’s. But we recommend the former since it uses python which we feel is a better first language to learn. CS50 is based on C (also covers python to some extent), is longer, and also covers other things like web programming etc. These areas are covered later as independent modules in this course, so we felt ok to skip that part. But motivated ones should certainly do CS50 in addition to the MIT one form here: <https://www.edx.org/course/cs50s-introduction-computer-science-harvardx-cs50x>

## Quick Introduction To Other Programming Languages

Duration: 10 days.

A lot of code you will encounter in different sections of this course, as well as your career in general, will span multiple programming languages. Hence it is important to develop familiarity with some. Though a large part is common across languages, there are some peculiarities of each.

Before going into further details of other languages, take an overview of different programming paradigms, or ways of programming. See: <http://cs.lmu.edu/~ray/notes/paradigms/>. Discussing all these in detail is not necessary for this course, so we will go through some commonly encountered languages. Nowadays languages are structured by mix-matching features of multiple paradigms.

### Java

Java is the most popular language used in software industry as of this writing. Go through this hands-on tutorial on Java for Python programmers: <http://interactivepython.org/runestone/static/java4python/Java4Python.html>

### C/C++

C is a core language one should generally understand, even though one might not be an expert in it. Many systems programs and explanations are provided in C, and knowing pieces like pointers help understand how low level code works with memory. C++ adds Object Oriented Programming features to C, and can be considered a superset of C. Quickly go through this booklet to get a hang of C++: <http://cs.slu.edu/~goldwamh/publications/python2cpp.pdf>

Dynamically allocated memory in C++ uses the new operator, while C does not have a new operator, has the malloc (and a couple of equivalent) function instead. Just know this: <https://www.pixelstech.net/article/1340193129-malloc-free-and-new-delete-in-C%2B%2B>

Since pointers, references, and dynamic memory allocation are very important concepts in C/C++ and confusing at first, it will help to study those in more detail. <http://cslibrary.stanford.edu/102/PointersAndMemory.pdf>.

Contrast dynamic memory allocation across Java (Python is similar to Java) and, C++. See: <https://www.ijecs.in/index.php/ijecs/article/download/1053/951/>.

### Web: HTML, CSS Javascript

HTML is a markup language used to design web pages. CSS is used to provide styling for the web pages. Javascript is a programming language typically used to manipulate HTML and CSS dynamically within the browser. And nowadays, Javascript is also used as a full-fledged programming language like Python.

Lear HTML and CSS from: <https://www.khanacademy.org/computing/computer-programming/html-css>

This quick talk on javascript for python programmers will serve as a succinct introduction to the language: <https://www.youtube.com/watch?v=GAoheEUiwwY>.

## Writing Good Code

Just writing programs is not enough. There are many more facets to it. It should be clean, performant, solid. Go through this article on writing good code: <https://www.thecodingdelight.com/writing-good-code/>

Once written, you should be in a position to debug it and troubleshoot. Go through: <https://simpleprogrammer.com/effective-debugging/>.

Go through this article on testing code, and apply learnings to all code you write. It uses C to demonstrate important principles, but those also apply to other programming languages. See: <https://www.cs.dartmouth.edu/~campbell/cs50/artoftesting.html>

Test automation is an inherent part of programming. Used to test various scenarios efficiently, as well as to ensure new code changes don’t break old functionality. Go through this article on automating tests for python code: <https://powerfulpython.com/blog/automated-tests-types-for-python/>

More on unit testing in Python: <https://jeffknupp.com/blog/2013/12/09/improve-your-python-understanding-unit-testing/>

Code should be efficient. Get a sense of various aspects of program optimization from this article: <https://en.wikibooks.org/wiki/Optimizing_Code_for_Speed>. Optimization is a vast topic. The references section has more links on this.

Logging is a very important aspect of any software. Useful not only to convey status, but also for troubleshooting. Understand effective Logging: <https://fangpenlin.com/posts/2012/08/26/good-logging-practice-in-python/>.

Commenting code: <https://levelup.gitconnected.com/comment-your-code-with-care-e355f3de2b34>.

Understand code reviews: <https://smartbear.com/learn/code-review/what-is-code-review/>, and code review best practices: <https://medium.com/palantir/code-review-best-practices-19e02780015f>, and <https://dzone.com/articles/dont-waste-time-code-reviews>.

Want to be among the best programmers? <https://hackernoon.com/how-to-become-the-best-programmer-in-the-world-ef9f584c81fa>

### Test

Attempt programming problems at: <https://www.hackerrank.com/domains/python> (Python), or <https://www.hackerrank.com/domains/java> (Java). Go through all, and attempt at least 2-3 across all difficulty levels and all areas you have learned above. Write solid code, test various boundary conditions.

## Theoretical Foundations

Duration: 10 days.

### Mathematics

First, go through this basic calculus course covering Limits, Derivatives, Integration, and Sequences and Series, from: <https://www.khanacademy.org/math/calculus-home> if you are not already comfortable with it.

Then do through this course on foundations of Mathematics for Computer science. You might choose to skip some assignments if you want, but understand the lectures well: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-spring-2015/index.htm>. Lecture videos are also available at this youtube playlist: <https://www.youtube.com/playlist?list=PLUl4u3cNGP60UlabZBeeqOuoLuj_KNphQ>.

This concise tutorial is a condensed version of the above two courses, but good enough only to get an overview. If you are not able to complete the above, at least go through this before moving ahead, and continue the above course whenever time permits. It is: <https://www.tutorialspoint.com/discrete_mathematics/index.htm>.

### Overview of Automata Theory

Automata Theory is a branch of computer science that deals with designing abstract computing devices that follow a predetermined sequence of operations automatically. An automaton is a machine which performs a range of functions according to a predetermined set of coded instructions.

This concise tutorial gives a good feel of this vast and complex area of computer science. It discusses Finite Automata, Regular Languages, and Pushdown Automata before moving onto Turing machines and Decidability: <https://www.tutorialspoint.com/automata_theory/index.htm>.

## Data Structures and Algorithms

Duration: 30 days.

This topic, along with programming, can be considered to be among the most important areas of computer science, its heart. Databases, Networks, AI, everything, everything else needs these. So take your time to do it thoroughly, it will help you throughout your career. There might be times of pain, frustration, since some topics need effort to get comfortable with. But hang on, be patient, take your time, take breaks as needed.

Do this Algorithms course from MIT. See videos, notes, and complete exercises and assignments: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/>

Try to finish this right through the end. But just in case you are too drained, at least try to finish this upto 75-80%, the initial part of Dynamic Programming. If needed, take a break, go through further topics, then revisit the remaining modules.

### Test

Attempt problems in the course material religiously.

Confirm you know various algorithm types as described here: <http://cds.iisc.ac.in/wp-content/uploads/DS221.L5.Algo_.Types_.pdf>

Try a few problems from recent ACM programming contests. Solving some of these will boost your problem solving and programming confidence like anything: <https://icpc.baylor.edu/worldfinals/problems>.

## Under the Hood: Computer Architecture, Systems Programs, Operating Systems

Duration: 10 days.

Before going into various related areas of Computing, it will help to know a few low-level topics that give insight into how things work inside the computer. These areas cover parts of computer hardware such as microprocessors, memories, and caches. System software such as compilers and linkers, various low level instruction sets, and structure of executable code. Another key area is Operating systems, that give users an abstract and easy interface to deal with various low level pieces.

Go through the small eBook ‘Bottom Up Computer Science from here: <https://www.bottomupcs.com/>. Though the title mentions Computer Science, it is not actually about theoretical computer science such as discrete maths, algorithms, etc. Instead it focuses on system level stuff such as architecture, compilers and others.

Then, study more operating systems basics from: <https://www.tutorialspoint.com/operating_system/index.htm>. Additionally, read about thread/process synchronization from: <https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_Synchronization.html>.

### Test

Answer these short quizzes:

Architecture: <https://quizizz.com/admin/quiz/568e4b9d9ddf8f0c686382ab/mrs-os-computer-architecture-quiz>

Operating Systems: <https://www.indiabix.com/computer-science/operating-systems-concepts/> and <https://medium.com/cracking-the-data-science-interview/the-10-operating-system-concepts-software-developers-need-to-remember-480d0734d710>.

## Commonly Used Programming Techniques

Duration: 10 days.

This section contains supplementary concepts very commonly needed in day-to-day programming. As you gain more and more experience with programming, these concepts will seep in better, but it is still important to develop an understanding and appreciation of those, have information of what is handy, in the initial learning stage itself.

The section does not have comprehensive exercises, but we strongly recommend executing code, experimenting, general playing around, and improve your insight into these important areas.

### Compiled, Interpreted, Just-In-Time Compiled Languages

A computer does not directly understand high-level languages like Python. It understands only machine language (i.e. *native or machine code*). A high-level program must therefore be translated into machine language so that the computer can execute it. The translation process varies according to the nature of programming language. This is explained in the links below:

<https://www.upwork.com/hiring/development/the-basics-of-compiled-languages-interpreted-languages-and-just-in-time-compilers/> and <http://www2.hawaii.edu/~takebaya/ics111/process_of_programming/process_of_programming.html>

### Useful Libraries/Modules/Packages

Understand the following features within python. Try out sample code to get a feel of those. No need to remember all features in detail, refer to documentation when need arises. We will study Networking, Databases and Multithreading in later sections. For now, just get a feel of what all features are typically available to programmers through Libraries. See: <https://www.tutorialspoint.com/python/index.htm> (Python), or <https://www.tutorialspoint.com/java/index.htm> (Java). Check out stuff like:

* Packages/Modules
* File IO
* Exceptions,
* Regular Expressions
* Database Access
* Networking
* Sending email
* Multithreading
* Key markup formats like XML, JSON, YAML: <https://kontrolissues.net/2015/07/28/xml-json-and-yaml-oh-my/>

### Annotations and Decorators

Annotations are a mechanism to add metadata to classes, functions and arguments, which in turn are used to build interesting language features like *Dependency Injection* and *Aspect Oriented Programming*.

You will encounter Annotations frequently while going through modern code, and will also use it often.

**Python:**

Decorators and Annotations: <https://code.tutsplus.com/tutorials/deep-dive-into-python-decorators--cms-29725>

Metaprogramming: <https://www.ibm.com/developerworks/library/ba-metaprogramming-python/index.html>

**Java:**

<https://howtodoinjava.com/core-java/annotations/complete-java-annotations-tutorial/>,

Dependency Injection using Google Juice (Python style Decoration): <https://www.baeldung.com/guice>

### Unicode, Internationalization and Localization

Everything about text is not just ASCII characters, where each character is 8 bytes. In today’s times, where applications and websites have to support multiple languages, one cannot afford not knowing about Unicode, Internationalization-Localization of software.

Go through these articles explaining what unicode is. Some examples use C language, but should be ok, you will understand: <https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/>, and <https://betterexplained.com/articles/unicode/>

Understand locales, internationalization and localization: <https://required.com/en/internationalization-localization/>,

See how to use Java or Python for Internationalization and Localization. E.g. Handle multi-language string messages, date-currency format variations across Locales, etc:

* Python: Locales: <https://pymotw.com/3/locale/>, and Text translation: <https://www.mattlayman.com/blog/2015/i18n/> (Python)
* Java: <https://dzone.com/articles/a-beginners-guide-to-java-internationalization> (Java)

### Concurrency, Parallelism, and Asynchronous IO

Parallelism is concerned with utilizing multiple processors/cores to improve the performance of a computation, generally using multiple threads or processes. Concurrency on the other hand, is concerned with managing access to shared state from different threads. Concurrency and parallelism are often confused terms, go through this to understand the differences: <https://blog.golang.org/concurrency-is-not-parallelism>. And either of these (Java or Python).

**Python:**

Parallel and concurrent programming in Python: <https://code.tutsplus.com/articles/introduction-to-parallel-and-concurrent-programming-in-python--cms-28612>,

Threadpools and Concurrent futures: <https://pymotw.com/3/concurrent.futures/>

Asynchronous programming in Python 3.4+ using async/await: <https://hackernoon.com/asynchronous-python-45df84b82434>, and <https://snarky.ca/how-the-heck-does-async-await-work-in-python-3-5/>

**Or, for Java folks**: <http://www.vogella.com/tutorials/JavaConcurrency/article.html>

# CS: Commonly Needed Topics

This section contains those topics from key areas of computer science which a typical software professional encounters pretty frequently. These are parts of subject areas such as Databases, Networks, Graphics, Compilers, and many others. The level of detail presented is based on what a typical professional needs in regular work life. Some topics like databases and networks in more details, overview of others. The references section has pointers to very detailed study of these subject areas. You could learn those afterwards gradually, or whenever need arises.

## Data Communications and Computer Networks

Duration: 15 days.

Understand basic concepts of communication and networking from these short tutorials: <https://www.tutorialspoint.com/communication_technologies/index.htm>, <https://www.tutorialspoint.com/principles_of_communication/index.htm>, and <https://www.tutorialspoint.com/data_communication_computer_network/index.htm>

Understand the Http protocol in sufficient detail. You will need it often: <https://www.tutorialspoint.com/http/index.htm>.

Get a hang of understanding RFCs through: <https://www.lifewire.com/what-is-internet-request-for-comments-rfc-4092366>. Then understand congestion control mechanisms through this: <http://ecomputernotes.com/computernetworkingnotes/communication-networks/what-is-congestion-control-describe-the-congestion-control-algorithm-commonly-used>. Now study the congestion control RFC: <https://tools.ietf.org/html/rfc5681>. Correlate with the explanation.

Understand Network Security. Before that, learn key concepts of Cryptography: See <https://www.tutorialspoint.com/cryptography/cryptography_quick_guide.htm>, and then <https://www.tutorialspoint.com/network_security/network_security_quick_guide.htm>.

Understand Digital Cetificates: <https://technet.microsoft.com/en-us/library/bb123848(v=exchg.65).aspx>

Get comfortable using SSH, see network security in action: <https://www.digitalocean.com/community/tutorials/ssh-essentials-working-with-ssh-servers-clients-and-keys>

Get used to some hands-on network programming from the following. As needed, see explanation of library functions by looking at the documentation of those from the internet.

* Python:
  + Sockets,TCP, UDP: <https://pymotw.com/2/socket/index.html>
  + HTTP: <https://pymotw.com/2/BaseHTTPServer/index.html> (Server), and <https://pymotw.com/2/urllib2/index.html> (Client)
  + XML-RPC (<https://en.wikipedia.org/wiki/XML-RPC>) Server: <https://pymotw.com/2/SimpleXMLRPCServer/index.html>
* Or Java: <http://www.buyya.com/java/Chapter13.pdf>.

## Database Systems

Duration: 30 days.

Databases Systems bring many areas of computing together. Right from Hardware, Algorithms, OS, Systems-Programming, and Networking. Secondly, irrespective of what area one works in, there is generally a lot of data to be managed and stored, so databases are used everywhere. Considering this, studying Databases in detail has many long term benefits. Deligently go through the lecture videos and do the homework assignments corresponding to this database course from CMU: <https://15445.courses.cs.cmu.edu/fall2017/index.html>

In addition to the above, go through the topic Text Search, from UCBs CS 186 course. The slides corresponding to the videos are available at: <https://sites.google.com/site/cs186spring2015/home/schedule-and-notes>, Videos are at: <https://www.youtube.com/watch?v=TrJYpayaLUw> and first half of <https://www.youtube.com/watch?v=ewpVoK2DwZs>

Get your hands more dirty actually working on a DBMS, namely MySQL. Go through these links and exercise various features, and attempt exercises: <http://www.ntu.edu.sg/home/ehchua/programming/sql/MySQL_HowTo.html> and <http://www.ntu.edu.sg/home/ehchua/programming/sql/mysql_beginner.html>

Python’s DB-API interface allows rich types of access to databases. See <https://docs.python.org/3/library/sqlite3.html> for the Sqlite implementation of DB-API. Other databases also have equivalent packages.

Understand Object relational mapping (ORM), which is used extensively for fast development, especially web applications. Learn this using SQLAlchemy package in Python: <https://docs.sqlalchemy.org/en/latest/orm/tutorial.html>.

Understand database performance optimization techniques, query plans, from: <https://msdn.microsoft.com/en-us/library/aa226166(v=sql.70).aspx>, and <https://www.red-gate.com/simple-talk/sql/performance/why-developers-need-to-understand-execution-plans/>

## System Analysis Design, Software Engineering

Duration: 15 days.

Systems Analysis and Design is an active field in which analysts repetitively learn new approaches and different techniques for building the system more effectively and efficiently. The primary objective of systems analysis and design is to improve organizational systems. Go through this tutorial to understand key concepts, : <https://www.tutorialspoint.com/system_analysis_and_design/index.htm>

Software engineering is an engineering branch associated with development of software products using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product. Get an overview of this area by going through: <https://www.tutorialspoint.com/software_engineering/index.htm>. Also check your understanding by attempting some exam and interview questions therein.

Get an overview of Object Oriented Design through this multi-part tutorial: <https://medium.com/omarelgabrys-blog/object-oriented-analysis-and-design-introduction-part-1-a93b0ca69d36>

Understand useful design patterns: <http://www.thedevpiece.com/design-patterns-that-every-developer-should-know/>

Understand Software Architectural patterns: <https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013>

About documentation (other than comments in code): Writing User stories: <https://blog.craft.io/2017/04/23/writing-epic-user-stories/> and <https://uxplanet.org/aligning-design-to-user-stories-614b4845fc8d>, Writing Requirements specs: <http://www.informit.com/articles/article.aspx?p=1152528&seqNum=4>, Writing Design docs: <https://medium.freecodecamp.org/how-to-write-a-good-software-design-document-66fcf019569c>, Writing Technical articles: <https://www.red-gate.com/simple-talk/how-to-write-an-interesting-technical-article/>. Writing test plans: <https://medium.com/@TheAtlasTeam/how-to-write-a-software-testing-plan-document-c47497dc4a05>. Formatting pages Using Markdown <https://www.markdowntutorial.com/>.

### Software Quality

We stressed upon the importance of testing code in the section on programming. In this section, we will elaborate further aspects of software quality.

This tutorial will introduce types of testing, methods, tools, and documentation: <https://www.tutorialspoint.com/software_testing/index.htm>.

Software quality factors to remember: <https://dzone.com/articles/10-groups-software-quality>. Understand unit, integration, and other types of tests in more detail: <https://huddle.eurostarsoftwaretesting.com/types-of-software-testing/>.

## Overview of Other Important Areas of Computing

Duration: 15 days.

Go through these short high-level tutorials to get a feel of other areas of computing. There are ample references to detailed courses on these areas in the last chapter. Learners should continue studying those depending upon areas in which their careers are shaping up.

Computer Graphics: <http://people.csail.mit.edu/fredo/Depiction/1_Introduction/reviewGraphics.pdf>, and <https://www.tutorialspoint.com/computer_graphics/index.htm>

Artificial Intelligence: See this video on AI being new Electricity <https://www.youtube.com/watch?v=21EiKfQYZXc>, and <https://www.tutorialspoint.com/artificial_intelligence/index.htm>.

Functional Programming: Part 1 <https://medium.com/@cscalfani/so-you-want-to-be-a-functional-programmer-part-1-1f15e387e536>, and Part 2 <https://medium.com/@cscalfani/so-you-want-to-be-a-functional-programmer-part-2-7005682cec4a>.

Compilers: <https://medium.com/@CanHasCommunism/understanding-compilers-for-humans-ba970e045877>, and <https://www.tutorialspoint.com/compiler_design/index.htm>.

Cloud Computing: <https://www.tutorialspoint.com/cloud_computing/index.htm>.

Embedded Systems: <https://www.tutorialspoint.com/embedded_systems/index.htm>

Internet of Things (IOT): <https://www.tutorialspoint.com/internet_of_things/index.htm>

Digital Communications. The quick guide at: <https://www.tutorialspoint.com/digital_communication/digital_communication_quick_guide.htm>,

Digital Image Processing. The quick guide at: <https://www.tutorialspoint.com/dip/dip_quick_guide.htm>

# Design and Development Industry Practices

Duration: 20 days.

In previous sections, we have covered various academic, and more fundamental areas of computing. When it comes to working in real life situations there are many other considerations. Without these, projects lacks rigor and finish necessary for any product/service that is being used for serious purposes. Be it business, scientific, or any other. Some of these are technical, some semi or non-technical. This section focuses on such general skills applicable to information technology. General purpose skills that have a broader applicability are covered in the Soft skills and life skills section.

Previously, development, testing, production deployment and monitoring jobs were managed by separate teams. Nowadays, due to the Agile nature of software development, many of these responsibilities are shared across roles of people on a project. Every role, and developers in particular, need to be well aware of test, deployment, and monitoring, and their automation, throughout the development cycle.

## Projects, Management Methodologies, Agile, Best Practices

Understand typical roles in software projects: <https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/> .

Understand basics of various project management methodologies and how they compare with each other: <https://medium.com/paymo/project-management-methodologies-the-complete-guide-fba376ef288a>.

Considering the dynamic nature of today’s software development and businesses in general, most software projects prefer an Agile development methodology. Understand the core four values and twelve basic principles of agile through: <https://www.smartsheet.com/comprehensive-guide-values-principles-agile-manifesto>.

Understand how Agile development compares with the traditional Waterfall model: <https://www.seguetech.com/waterfall-vs-agile-methodology/>.

Understand Scrum Agile methodology in more detail. It is the most popular among all agile methodologies. See: <https://plan.io/blog/scrum-a-collection-of-some-of-the-best-resources/>, and then, consolidate your understanding by going through the official Scrum guide: <https://www.scrumguides.org/scrum-guide.html>. After that, this practical overview of Scrum: <https://hackernoon.com/a-practical-scrum-overview-f46810295e8b>, and understand best practices of various Scrum processes in further detail (E.g. Standups: <https://www.solstice.com/fwd/top-10-best-practices-for-productive-stand-ups>, Spring Planning: <https://www.yodiz.com/blog/agile-methodology-and-sprint-planning-rules-and-best-practices/>, Retrospective: https://thedigitalprojectmanager.com/how-run-sprint-retrospective/ )

Get a rough feel of various Agile development ‘Tribes’, areas of concern, and relations between them: <https://www.agilealliance.org/agile101/subway-map-to-agile-practices/>. It has links to glossary of various terms.

Test Driven Development methodology of development involves writing automated tests before writing code. The overall result is more robust code, that is well in sync with requirements. Get an overview from: <https://medium.freecodecamp.org/test-dritven-development-what-it-is-and-what-it-is-not-41fa6bca02a2>.

About executing software projects successfully: <https://www.codeproject.com/Articles/32721/The-7-Elements-of-Highly-Successful-Software-Proje> and <https://ac.els-cdn.com/S2212017312004537/1-s2.0-S2212017312004537-main.pdf?_tid=d3428454-9e83-41d4-976d-593b798d9e82&acdnat=1535391196_edc1d0aef7c21aa1a79a80cc1a496d7c>

More project management activities: Effective meetings: <https://www.businessinsider.in/Google-Chair-Eric-Schmidts-8-Rules-For-Running-A-Great-Meeting/articleshow/43913849.cms> and <https://medium.com/swlh/meetings-d27a3c74e5f5>, Minutes of meeting: <https://journals.lww.com/nursingmanagement/fulltext/2013/03000/Documenting_the_minutes_at_professional_meetings.11.aspx>, Status reports: <https://business.tutsplus.com/tutorials/how-to-write-better-status-reports--cms-23479>,

Experiences on delivering projects with tight deadlines: <https://remote.online/insights/delivering-a-big-project-to-a-tight-deadline>.

## Management, Design and Development Tools

This section contains some commonly used tools used in various Planning, Design and Development activities over the software lifecycle.

### Editors and IDE

It is always useful to have a well-featured lightweight editor handy. VSCode is one such promising editor: <https://flaviocopes.com/vscode/>, which is also an integrated development environment in its own right.

Some important IDEs used extensively are:

Eclipse: <http://www.vogella.com/tutorials/Eclipse/article.html>, Debugging with Eclipse: <https://www.eclipse.org/community/eclipse_newsletter/2017/june/article1.php>

Idea Intellij: <https://www.jetbrains.com/idea/documentation/>, Debugging using Intellij: <https://www.jetbrains.com/help/idea/debugging-code.html>

Debugging in VSCode: <https://code.visualstudio.com/docs/editor/debugging>

### Source Code Control (Git)

Read the first three chapters of this book, upto Git Branching, to understand the basics of version control, and Git: <https://git-scm.com/book/en/v2>.

Learn how to use git by going through this interactive tutorial. You could skip the Advanced Topics section if you want: <https://learngitbranching.js.org/>

Go through this article to understand good practices on branching: <https://nvie.com/posts/a-successful-git-branching-model/>

### Build Tools

Understand what a software build involves: <http://scmquest.com/software-build-knowledge/>. Get to know popular build tools:

* Make <http://mrbook.org/blog/tutorials/make/> and <https://felixcrux.com/blog/creating-basic-makefile>, mainly for C and C++ code. For cross platform make, cmake, see: <https://cmake.org/cmake-tutorial/>.
* Gradle, used extensively for Java projects: <http://tutorials.jenkov.com/gradle/gradle-tutorial.html>

Build tools that trigger the build whenever necessary repository changes are detected, are described later in the Continuous Delivery section.

### Code Analysis

Code reviews using Google Gerrit: <http://www.vogella.com/tutorials/Gerrit/article.html>.

Understand Code Coverage: [https://www.atlassian.com/continuous-delivery/introduction-to-code-coverag.e](https://www.atlassian.com/continuous-delivery/introduction-to-code-coverage), and code coverage in Python: <https://coverage.readthedocs.io/en/coverage-4.5.1a/>

Understand Code profiling, to find bottlenecks and improve performance (in python): <https://www.integralist.co.uk/posts/profiling-python/> and <https://devopedia.org/profiling-python-code>.

There are many tools that do static analysis of code to check style, code smell, find potential bugs. See: <https://www.codacy.com/blog/review-of-python-static-analysis-tools/>. SonarQube is another one: <https://dzone.com/articles/why-sonarqube-1>

### Project Management

All activities of software development need to be managed using appropriate tools. These range from project management tools like those for scrum and other agile styles, wikis for collaboration and documentation, integration with code repositories, code review, test automation, defect tracking, and continuous build systems.

Jira is one very popular tool that covers all the above aspects. A free alternative is Tuleap. See <https://www.tuleap.org/what-is-tuleap>. Understand different parts of this platform from: <https://www.tuleap.org/resources/videos-tutorials>.

### General Purpose Tools

#### The Spreadsheet

Like Unix Shell, spreadsheets is another useful thing that will help you crunch data in many different ways, throughout your technical career. A data organization tool at heart, yet, a kind of general purpose tool used by many roles for many different end uses.

Understand how to use spreadsheets at a high level. Either MS-Excel, Open/LibreOffice Excel, or Google Docs. The following links use Google Spreadsheets as the spreadsheet, but the same principles apply to others like Microsoft Excel or Openoffice spreadsheets.

* Learn spreadsheet basics from this video: <https://www.youtube.com/watch?v=M3-MDhnZzPU>
* Try out various features of spreadsheets described here: <https://gsuite.google.com/learning-center/products/sheets/get-started/#!/>

Finally, take this quick quiz on Spreadsheets (Google): <https://quizizz.com/admin/quiz/58c0c6442f772c1d18276ec9/google-sheets>

#### General Tools

Search google to know about the some useful utilities and be aware of what they can be useful for.

Network: Curl, Netcat (nc)

Packet analysis tools: Ethereal, Fiddler, etc.

Monitoring and Troubleshooting: iostat, netstat, vmstat, top, strace

Collaboration: Skype, Slack, Google Hangouts, Facebook

Package management tools: (Ubuntu) apt

## DevOps, Continuous Integration and delivery, Monitoring

Understand what devops, a very important aspect of modern software deployments: <https://www.versionone.com/devops-101/what-is-devops/>

The DevOps tool landscape: <https://devup.co/a-look-at-devops-tools-landscape-7220099c6b81>.

### Capacity Planning

Capacity planning is an important exercise to be done before deploying applications. See this for what it involves, how it is done: <https://wso2.com/whitepapers/capacity-planning-for-application-design-part-1/>.

### Continuous Integration

Go through this article to understand continuous integration: <https://codeship.com/continuous-integration-essentials>

Continuous integration with Jenkins: <http://www.vogella.com/tutorials/Jenkins/article.html>

### Continuous Deployment and Delivery

Understand the concept of Infrastructure as Code: <https://www.thoughtworks.com/insights/blog/infrastructure-code-reason-smile>

Ansible is an open source automation platform to help with configuration management, application deployment, task automation. <https://serversforhackers.com/s/ansible>. It prepares machines corresponding to a cluster.

Understand containers and Docker: <https://www.youtube.com/watch?v=YFl2mCHdv24> and <https://serversforhackers.com/c/getting-started-with-docker>

Managing (pushing out) docker containers with Ansible: <https://linuxacademy.com/howtoguides/posts/show/topic/13750-managing-docker-containers-with-ansible>

Understand container orcherestration with Kubernetes (responsible to keep docker containers up and running): <https://www.infoworld.com/article/3268073/kubernetes/what-is-kubernetes-container-orchestration-explained.html>

### Metrics Collection, Monitoring, Alerting

Introduction to metrics, monitoring and alerting: <https://www.digitalocean.com/community/tutorials/an-introduction-to-metrics-monitoring-and-alerting>.

Tools for metrics collection, monitoring and alerting.

* Log collection, monitoring and analysis through ELK stack: <https://fangpenlin.com/posts/2012/08/26/good-logging-practice-in-python/>.
* Metrics Collection and export for further analysis using Prometheus client: <https://github.com/prometheus/client_python>.
* Metrics Monitoring and Alerting using Prometheus: <https://prometheus.io/docs/introduction/overview/> and <https://prometheus.io/docs/prometheus/latest/getting_started/>.
* Metrics Visualization using Graphana: <https://www.youtube.com/watch?v=sKNZMtoSHN4&index=7&list=PLDGkOdUX1Ujo3wHw9-z5Vo12YLqXRjzg2> and <http://docs.grafana.org/guides/getting_started/>.

# Electives: Focussed Technology Tracks

Duration: 30-60 days per track.

Now that you have a reasonable hold on the basics of software development as well as a sense of important industry practices, it is time to learn a few areas at greater depth. Doing this will not only make you a skilled worker in those areas, but also give a sense of how to proceed when confronted with an unknown topic that you need to ramp up on. Go through a few areas your choice and gain expertise over those.

We will continue the work of compiling quality learning resources, but while that happens, you should also develop the ability to search what all it takes to gain expertise. To find quality learning material from wherever possible, learn stuff in sufficient detail, and be ready to deliver like a Pro. Once this ability is developed, the fear of getting outdated due to fast changing technology need not bother you ever. You know what to do.

While you go through those tracks in more detail, also select a project that you will be doing in those technologies. Start thinking about how you will design and implement the project, while you are going through the technology track(s).

## Multi-tier Web Application Backend Development

Duration: 30 days.

This track covers development of application backends that can be used in traditional web applications, modern browser-rich applications, or native applications needing support of common, shared, services.

Understand Web Applications: <https://lunarpages.com/understanding-web-applications/>

Get introduced to Server side programming: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps>

The following sections demonstrate web application development using Python, and within it, the frameworks Flask and Django, However the principles remain similar whether it is .Net, or Java-Spring/SpringBoot, or something else. Only the specifics will change.

### Building Web Applications

Understand how Python is used in web servers: <https://docs.python.org/3.4/howto/webservers.html>[l](https://www2.dmst.aueb.gr/dds/pubs/conf/2002-SANE-DynCont/html/dyncont.html)

Get more hands-on with Server side web programming using Python-Flask and all the related features through: <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>.

### Understanding Backend Architectures

N-Tier Architectures and Tips: <https://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips>

Architectural shift in web applications: <https://dzone.com/articles/architectural-shift-in-web-applications-with-emerg>

Service Oriented Architectures <https://www.ibm.com/support/knowledgecenter/en/SSMQ79_9.5.1/com.ibm.egl.pg.doc/topics/pegl_serv_overview.html>

Microservice Architectures: <https://martinfowler.com/articles/microservices.html>

Serverless Architectures: <https://martinfowler.com/articles/serverless.html>

### Building Applications using MVC, Microservices

Get the main intuition behind the model view controller architecture in typical web applications: Understand MVC <https://medium.freecodecamp.org/model-view-controller-mvc-explained-through-ordering-drinks-at-the-bar-efcba6255053>

MVC applications using Python. This time, the web framework used is Django: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django>

Understand more about web services in Python: <https://medium.com/@umerfarooq_26378/web-services-in-python-ef81a9067aaf>

Understand how microservices are built in Python: <https://www.toptal.com/python/introduction-python-microservices-nameko>

### Authentication, Authorization, Security

Authentication, Authorization, and in general, security is crucial to shared backends.

Understand the basics of web application security: <https://martinfowler.com/articles/web-security-basics.html>.

Understand various Web Authentication Methods: <https://blog.risingstack.com/web-authentication-methods-explained/>

Understand various ways of Single Signon: <https://www.marcelbelmont.com/post/concepts-in-single-sign-on-using-jwt-and-saml/>

Authentication using Flask and OAuth: <https://blog.miguelgrinberg.com/post/oauth-authentication-with-flask>

Authorization and role based access using Python-Flask: <http://blog.tecladocode.com/learn-python-defining-user-access-roles-in-flask/>

Web Application Security in Flask: <http://flask.pocoo.org/docs/1.0/security/>

Avoiding Brute force attacks in Flask apps with reCaptcha: <https://techmonger.github.io/5/python-flask-recaptcha/>

Running Python-Flask web application over HTTPS: <https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https>

### Software As A Service (SAAS) Applications

Modern software uses a single deployment of software to serve multiple organizations and different classes of users within them.

Understand special considerations when a single application is used to serve multiple tenants. <https://www.cmg.org/wp-content/uploads/2012/11/m_94_4.pdf>

See this talk on road to building SAAS applications with Python: <https://www.infoq.com/presentations/saas-python>

Study the source code of this Flask based SAAS Game app: <https://github.com/nickjj/build-a-saas-app-with-flask>. There is little documentation, you need to go through the code and understand it.

### Performance and Scalability

Go through this comprehensive article on web application performance and scalability: <https://www.webforefront.com/performance/index.html>

Look at this guide on optimization of django based web apps: <https://www.toptal.com/python/performance-optimization-testing-django>

Understand caching in Flask: <http://brunorocha.org/python/flask/using-flask-cache.html>

Understand Load Balancing Flask with HAProxy: <https://www.vioan.eu/blog/2016/06/24/scale-your-python-app-with-docker-and-haproxy/>

### Asynchronous Operations

Understand asynchronous web applications, by studying the Tornado Framework used in IO intensive applications needing high concurrency: <http://www.tornadoweb.org/en/stable/guide.html>.

Now Understand WebSockets, server notifying the client (browser), and client not needing to poll server: <http://blog.teamtreehouse.com/an-introduction-to-websockets>, and implementing websockets in Tornado <https://os.mbed.com/cookbook/Websockets-Server>

Understand Asynchronous Messaging using Message Queues and Pub/Sub Queues: <https://aws.amazon.com/message-queue/>, <https://aws.amazon.com/pub-sub-messaging/>.

Understand how Asynchronous message is implemented with Celery and RabitMQ: <http://docs.celeryproject.org/en/latest/getting-started/first-steps-with-celery.html#tut-celery>, and using asynchronous task queues with Django: <https://medium.com/@ffreitasalves/executing-time-consuming-tasks-asynchronously-with-django-and-celery-8578eebab356>.

Understand task queues and publish/subscribe queues using Celery and RabbitMQ through this tutorial: <http://www.bogotobogo.com/python/RabbitMQ_Celery/python_Installing_RabbitMQ_Celery.php>.

### Production Deployments

See various options of deployment of a flask application from: <http://flask.pocoo.org/docs/1.0/deploying/>, in particular, the sections on uWSGI and FastCGI and a couple of deployments on cloud.

Understand deployment of flask based apps using uWSGI and Nginx: <https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-uwsgi-and-nginx-on-ubuntu-14-04>

Understand more about docker: <https://medium.freecodecamp.org/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b>.

Refresh your understanding of deploying flask web apps on a docker container: <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-xix-deployment-on-docker-containers>.

Get to know components in a serious deployment of a Django or Flask app: <https://stackoverflow.com/questions/13210636/differentiate-nginx-haproxy-varnish-and-uwsgi-gunicorn>. See them being deployed together by studying: <https://github.com/jvuori/flask-uwsgi-nginx-haproxy-docker>

### Secured Connections - HTTPS

For any serious web application, data going between a client and backend application server has to be encrypted. Read on, to understand more about adding support for such secure transfers.

About HTTPS and Digital Certificates: <https://www.kaspersky.com/blog/digital-certificates-https/1797/> and <https://www.instantssl.com/ssl-certificate-products/https.html>.

Setting up a Django/uwsgi web app for https: <https://stackoverflow.com/questions/29827299/django-uwsgi-nginx-ssl-request-for-working-configuration-emphasis-on-ss>

### Case Studies

Study this CRM tool based on Python-Django. Study documentation, run it, see source code, play around. Also understand how testing and deployment is handled. See: <https://github.com/MicroPyramid/Django-CRM>

## Browser Based Rich Applications with HTML-CSS-Javascript

Duration: 40 days.

The previous section focussed on multi-tier web application backends, and a bit on the browser side HTML/CSS UI. When the client is a browser, the backend serves static HTML, CSS, Javascript files as well as dynamic HTML content. However, modern applications do a lot more work on the front end side (browser or native mobile app). Thus, the backend acts almost like a data server (JSON, XML, web services), and almost no dynamic HTML content. Backends just return some html templates, CSS and javascript, images, etc. The rest is data as JSON, XML or others. The look and feel of these apps is almost like native apps, very lightweight and fast. These are the modern day Rich Internet applications, often, “Single Page Web Applications”, which we now describe in more detail.

Get an overview of Rich Internet Applications: <http://tutorials.jenkov.com/software-architecture/ria-architecture.html>.

Understand what Single Page Applications (SPAs) are, from: <https://blog.4psa.com/an-intro-into-single-page-applications-spa/>, and <https://www.seguetech.com/what-is-a-single-page-application/>.

Understand about cross-platform desktop application development using using Html/Css/JS: <https://medium.com/@paulbjensen/building-cross-platform-desktop-applications-with-electron-and-nw-js-e1fc93ae3bff>.

Understand Progressive apps, i.e. building apps using Html/Css/JS, that work within the browser locally: <https://www.howtogeek.com/342121/what-are-progressive-web-apps/>.

And then hybrid apps for mobiles and tablets: <https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/>.

### Core Tools: HTML(5), CSS(3), Javascript, SASS

The Rich UI applications are much more interactive and complex compared to traditional form based HTML apps. Almost like a native desktop application. Over time, many technologies have been tried for building such apps, such as Flash, Silverlight, Applets, and Javascript. But as of this writing, it looks pretty clear that Html/CSS/Javascript are here to stay. Be it Single Page Web Apps, Desktop Apps, Progressive Apps, or mobile Apps (using Cordova or similar technologies). And code structure for all types of these apps is similar to a huge extent.

Learn Basic HTML and CSS from: <https://learn.shayhowe.com/html-css/>,

Then learn, advanced HTML and CSS techniques from: <https://learn.shayhowe.com/html-css/>.

SASS (Syntactically Awesome Style Sheets) is an extension of CSS that enables you to use variables, nested rules, inline imports and more. It also helps to keep things organised and allows you to create style sheets faster. Understand SASS, <https://www.creativebloq.com/web-design/what-is-sass-111517618>, and then learn some key concepts from: <https://sass-lang.com/guide>

Learn Javascript in sufficient detail through: <http://javascript.info/>. If you have some Javascript background, go through book “Eloquent Javascript”: <https://eloquentjavascript.net/>.

JQuery is a very popular library used in Javascript, providing many utility functions available to programmers. Take a further overview of JQuery from: <http://jqfundamentals.com/>.

Typescript is one of the many attempts to make writing Javascript programs ‘better’. A wrapper over javascript which is eventually compiled to Javascript. Considering the probability of its use, it is useful to have a fair overview of it: <http://blog.teamtreehouse.com/getting-started-typescript>.

Learn about some more cool things that HTML/CSS/JS can do: Animations- <https://cloudinary.com/blog/creating_html5_animations>, Maps with Javascript- <https://www.tutorialspoint.com/leafletjs/index.htm>.

### Building Delightful Web Experiences

Go through this elaborate guide from experienced Google developers, regarding developing delightful web experiences. It is pretty elaborate, feel free to skip detailed descriptions of specific aspects, or revisit later, if you want to jump to building apps right away: <https://developers.google.com/web/fundamentals/>.

Explore sites with some great UI: <https://careerfoundry.com/en/blog/ui-design/8-sites-with-great-ui/>, <https://www.creativebloq.com/web-design/examples-ui-design-7133429>, and <https://medium.com/mockplus/10-best-app-ui-design-for-your-inspiration-in-2017-938db514053b>

Get used to wireframing with a free tool such as wireframe.cc: <https://www.youtube.com/watch?v=3dXsRNS-spc>, and <https://wireframe.cc/docs/>.

Bootstrap is among the most popular frontend component libraries for building good-looking applications. See: <https://getbootstrap.com/>, and go through the tutorial: <https://www.w3schools.com/bootstrap4/default.asp>

Understand Material Design, a design language by Google, initially used to build user interfaces in Android devices: <https://envato.com/blog/introduction-material-design/>.

### MVC on UI Side

Previously, since UI functionalities were limited, the amount of design that needed to go into building browser side interactivity was limited to widgets such as buttons, select boxes and forms, etc., with standard functionality for those widgets, and JQuery for more interactivity. But now, since a lot of functionality is implemented in the browser, it is also important to have patterns, frameworks, and tools to organize code on the browser side. This is similar to the MVC based architecture where data model, html view, and the piece that combines these two, the controller, were separated out.

Go through this article on need for Client side MVC like frameworks: <http://jster.net/blog/why-should-you-use-client-side-mvc-framework>.

Understand how MVC on client side is implemented in Pure Javascript: <https://www.sitepoint.com/mvc-design-pattern-javascript/>, or <https://alistapart.com/article/javascript-mvc>.

While there are many great frameworks to build Javascript based apps in a structured way, such as React, Angular, and Vue.js. This course uses React, but you can find equivalent articles for Angular or Vue from the web.

### Interactive user interfaces with React

React a javascript library for building interactive user interfaces, among the most popular framework to build HTML-CSS-JS based apps today.

Check out the homepage of React: <https://reactjs.org/>.

Go through this crash course video on React: <https://www.youtube.com/watch?v=Ke90Tje7VS0>.

Then go through this beginners tutorial: <https://www.taniarascia.com/getting-started-with-react/>.

And then these beginner-intermediate tutorials: <https://reactjs.org/docs/hello-world.html>, and <https://reactjs.org/tutorial/tutorial.html>.

Get an overview of advanced features by going through the initial part of those. Just to know what all is available. Study some more advanced features in more detail such as: Contexts (<https://reactjs.org/docs/context.html>) and Refs (<https://reactjs.org/docs/refs-and-the-dom.html>).

Get an overview of Jest, the powerful library to test javascript apps: <https://jestjs.io/> and <https://jestjs.io/docs/en/getting-started.html>.

Learn about testing React apps with Jest: <https://jestjs.io/docs/en/tutorial-react>.

Understand performance considerations in React Apps: <https://reactjs.org/docs/optimizing-performance.html>, <https://medium.com/@joomiguelcunha/react-performance-tips-5fa199a450b2>, and <https://medium.com/myheritage-engineering/how-to-greatly-improve-your-react-app-performance-e70f7cbbb5f6>.

Understand how material design can be done in React UIs through: <https://www.youtube.com/watch?v=PWadEeOuv5o>, and <https://material-ui.com/getting-started/installation/>.

### Structuring React Apps with Flux

Flux is a very popular application architecture from Facebook, used to structure modern client-side web apps. It is an improvement on the MVC architecture, considering the kind of interactivity that rich UI apps need.

First, understand the motivation behind flux and how it compares to the MVC architecture: through this in-depth overview: <https://facebook.github.io/flux/>

Understand basic Flux concepts: <https://github.com/facebook/flux/tree/master/examples/flux-concepts>

Go through this TODO MVC sample application in detail: <https://github.com/facebook/flux/tree/master/examples/flux-todomvc>

Redux is a predictable application state container for javascript that evolves the above idea of Flux. Go through an overview of the same: <https://redux.js.org/>, and <https://redux.js.org/introduction>.

Then go through these basic video tutorials: <https://egghead.io/series/getting-started-with-redux>, and <https://egghead.io/courses/building-react-applications-with-idiomatic-redux>. Learn it in a different way to understand better: <https://redux.js.org/basics>

Learn advanced concepts and get more hands on: <https://redux.js.org/advanced>, and <http://www.thegreatcodeadventure.com/building-a-simple-crud-app-with-react-redux-part-1/>.

Designing React-Redux Apps: <https://medium.freecodecamp.org/the-best-way-to-architect-your-redux-app-ad9bd16c8e2d>, <https://medium.com/javascript-scene/10-tips-for-better-redux-architecture-69250425af44>, <https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0>

Learn how to unit test Redux with Jest: <https://redux.js.org/recipes/writingtests>

### UI Performance

See articles in this section: <https://developers.google.com/web/fundamentals/performance/why-performance-matters/>.

### Build Tools

Go through this article to understand various build and bundler tools. Put them in context with respect to each other in the context of javascript code: <https://medium.freecodecamp.org/making-sense-of-front-end-build-tools-3a1b3a87043b>

Now understand how to build/package a react application using Webpack: <https://hackernoon.com/how-to-build-a-todo-app-using-react-redux-and-webpack-1aa99dc2f45c>

### Variations of Javascript Apps: Progressive, Native Desktop, Hybrid

Get an overview of building Progressive web apps with React: <https://medium.com/front-end-hacking/build-a-realtime-pwa-with-react-99e7b0fd3270>.

Get to know how to build Javascript based desktop apps using React: <https://medium.freecodecamp.org/building-an-electron-application-with-create-react-app-97945861647c>.

Hybrid apps is one way to build Mobile apps, perhaps not the most preferred one. Nevertheless, for web UI developers, this is a good way to do mobile application development. They need not study a new language end to end. See <https://www.toptal.com/mobile/developing-mobile-applications-with-apache-cordova> to understand how hybrid apps are built. And this one on building an hybrid app using React-Redux: <https://medium.com/the-web-tub/creating-a-cordova-hybrid-app-with-react-redux-and-webpack-13fe24b6b272>.

React-native is a framework to build native mobile apps using Javascript and React. Though we will not cover Mobile application development in this learning track, this is a good thing to be aware of: <https://facebook.github.io/react-native/docs/tutorial.html>.

### Case Study:

Cezerin is an eCommerce platform based on React and Redux. Go through the project in detail. Run it, see source code, modify: <https://github.com/cezerin/cezerin>.

## Machine Learning and Deep Learning

Duration: 60 days.

Before diving into Machine learning, get some Mathematical fundamentals in place by going through this course: <https://www.edx.org/course/essential-math-machine-learning-python>.

Do this excellent course on machine learning. Covers theory well, and also has hands-on exercises (audit is free): <https://www.coursera.org/learn/machine-learning/>.

Do deep learning specialization on Coursera (audit is free): <https://www.coursera.org/specializations/deep-learning>

Learn how to use Tensorflow in more detail from this course from Google: <https://www.udacity.com/course/intro-to-tensorflow-for-deep-learning--ud187>

Other References (Use discretion):

* Broader and forward looking aspects of general machine intelligence: Videos under <https://agi.mit.edu/>
* First do this machine learning crash course to understand fundamentals of tensor flow: <https://developers.google.com/machine-learning/crash-course/>
* Go through tutorials at: <https://www.tensorflow.org/tutorials/>
* Learn data Analysis with Python using Pandas: <https://pythonprogramming.net/data-analysis-python-pandas-tutorial-introduction/>.
* Machine Learning with Python: <https://pythonprogramming.net/machine-learning-tutorial-python-introduction/>.
* Go through building a chatbot with Python and Tensorflow: <https://pythonprogramming.net/chatbot-deep-learning-python-tensorflow/>
* Go through: <https://www.datacamp.com/community/open-courses/kaggle-python-tutorial-on-machine-learning>, about predicting the survival rate of Titanic disaster.

## Android Development

Learn Native Android App development.

First learn basics of User Experience design from: <https://classroom.udacity.com/courses/ud849>. And about material design for Android: <https://www.udacity.com/course/ud862>.

If you are not a reasonably good Java programmer yet, do this very basic Android development course, which will also get you more comfortable with Java: <https://in.udacity.com/course/android-basics-user-interface--ud834-india>.

Go through Google’s Android Development Fundamentals Course: <https://developer.android.com/courses/fundamentals-training/overview-v2>

Go through Google’s Advanced Android Development Course: <https://developer.android.com/courses/advanced-training/overview>.

Kotlin possibly could turn out to be the recommended Android app development platform. Get introduced to it: <https://in.udacity.com/course/kotlin-bootcamp-for-programmers--ud9011>.

## IOS Development

Get yourself an Apple machine for development. Even a Mac-mini will do.

Do this comprehensive course from Stanford (available only as an app on IOS): <https://itunes.apple.com/in/course/developing-ios-11-apps-with-swift/id1309275316>.

## Bigdata Technologies

### Overview

Bigdata, Concepts and Applications at: <https://data-flair.training/blogs/big-data-tutorials-home/>.

### Hadoop: HDFS and, Mapreduce

Hadoop: <https://data-flair.training/blogs/hadoop-tutorial/>

HDFS: <https://data-flair.training/blogs/hadoop-hdfs-tutorial/>.

Map Reduce: <https://data-flair.training/blogs/hadoop-mapreduce-tutorial/>.

### Data load

See Flume and Sqoop sections in tutorial: <https://data-flair.training/blogs/hadoop-tutorials-home/>.

### File formats: Avro, Protobuf

See Avro section in: <https://data-flair.training/blogs/hadoop-tutorials-home/>.

### Architecture: Data Lake, Toolset.

<https://medium.com/@rpradeepmenon/demystifying-data-lake-architecture-30cf4ac8aa07>

### Data Analysis

Hive section of <https://data-flair.training/blogs/hadoop-tutorials-home/>.

Pig section of <https://data-flair.training/blogs/hadoop-tutorials-home/>.

### Massive data storage for quick access: HBase

See HBase section of <https://data-flair.training/blogs/hadoop-tutorials-home/>.

Study Cassandra: <https://data-flair.training/blogs/cassandra-tutorials-home/>

Compare HBase and Cassandra.

### Messaging and Queues: Kafka

Beginner to Expert tutorials at: <https://data-flair.training/blogs/kafka-tutorials-home/>.

### Stream Processing and Analysis: Spark

3 levels of tutorials on spark from Beginner to Expert at: <https://data-flair.training/blogs/spark-tutorials-home/>.

### Job Scheduling: Oozie

See <http://oozie.apache.org/docs/5.0.0>.

### Cluster management

Zookeeper section of <https://data-flair.training/blogs/hadoop-tutorials-home/>

HCatalog section of <https://data-flair.training/blogs/hadoop-tutorials-home/>

Yarn section of <https://data-flair.training/blogs/hadoop-tutorials-home/>

Ambari section of <https://data-flair.training/blogs/hadoop-tutorials-home/>

## Cloud Computing (Work in progress)

This section is being enhanced

Cloud Computing: Beginner to Expert tutorials at <https://data-flair.training/blogs/cloud-computing-tutorials-home/>.

Amazon AWS Cloud: <https://data-flair.training/blogs/aws-tutorials-home/>.

Microsoft Azure: <https://www.tutorialspoint.com/microsoft_azure/index.htm> and <https://azure.microsoft.com/en-in/get-started/>

## Games, Augmented Reality, Virtual Reality (Work in Progress)

Section being updated.

AR/VR Overview: <https://in.udacity.com/course/introduction-to-virtual-reality--ud1012>.

Unity Engine: <https://www.edx.org/course/creating-virtual-reality-vr-apps> and/or <https://www.raywenderlich.com/unity>.

AR/VR using Vuforia: <https://library.vuforia.com/articles/Training/getting-started-with-vuforia-in-unity.html>.

Unreal Engine: <https://www.raywenderlich.com/unreal-engine>.

AR VR frameworks using javascript: <https://createwebvr.com/>

## IOT (Work in Progress)

This section is being enhanced.

IOT Concepts, Techniques, Applications: <https://data-flair.training/blogs/iot-tutorials-home/>

## Devops (Work in Progress)

This section is being enhanced.

Understanding Devops (6 part series): <https://sdarchitect.blog/2012/07/24/understanding-devops-part-1-defining-devops/>

Ansible: <https://serversforhackers.com/s/ansible>

Docker: <https://docker-curriculum.com/>

Kubernetes: <https://kubernetes.io/docs/tutorials/kubernetes-basics/>.

Mesos: <http://mesos.apache.org/getting-started/>.

CI/CD Jenkins: <https://dzone.com/articles/building-a-continuous-delivery-pipeline-using-jenk>.

Prometheus: <https://prometheus.io/docs/introduction/overview/>.

## More Technical Tracks (Horizontal Domains)

Like the previously elaborated technical tracks, there are many more used in the industry, and used across multiple business domains. Till we progress on compilation of learning material for many of these, you could try to put it together yourself by taking some effort using Google search, talking to knowledgeable people out there, and other means.

* QA and Test Automation
  + QA: <https://in.udacity.com/course/software-testing--cs258>.
  + Selenium: <https://www.seleniumhq.org/docs/01_introducing_selenium.jsp>, <https://www.seleniumhq.org/docs/03_webdriver.jsp>.
* Business Intelligence
* Digital transformation of small businesses and organizations using Open Source and Cloud technologies: CMS, CRM, ECommerce, Social Media, etc.
  + See: <https://www.coursera.org/learn/digital-transformations/home/welcome>.
* ...

## Domain Tracks

Like horizontal technical tracks, there are many different business verticals that need a lot of serious IT support. Till we progress on compilation of learning material for many of these, you could try to put it together yourself by taking some effort using Google search, talking to knowledgeable people out there, and other means.

Some example tracks are:

* Telecommunication
* Banking, Finance and Insurance
* Medical Sciences
* ...

# Special Topics For Detailed Study

Duration: 10 days.

Study at least 5 papers among different topics given below. Try to understand to the best of your ability. Wherever things look too difficult or mathematical, at least try to understand the intuition behind the math, keep that part separate (as not understood, for later study, discussions with others perhaps) and proceed ahead.

Before going through individual papers, go through Keshav’s paper on how to study a research paper: <http://ccr.sigcomm.org/online/files/p83-keshavA.pdf>

## Algorithms

Programming Parallel algorithms: <https://pdfs.semanticscholar.org/c176/ae8e58841713e3d9dc5e36cfabfaeac9749e.pdf>

R-trees for indexing spatial data: <https://pdfs.semanticscholar.org/eecc/40f0ad656db8191e0243ac10de9f4e36b4e3.pdf>

Mechanical Sympathy and the Disruptor: <http://lmax-exchange.github.io/disruptor/files/Disruptor-1.0.pdf>, and then <https://lmax-exchange.github.io/disruptor/>.

The Byzantine General’s problem, is an agreement protocol that’s built around an imaginary General who makes a decision to attack or retreat, and who must communicate his decision to his lieutenants <https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf>

Sort vs Hash revisited: <https://15721.courses.cs.cmu.edu/spring2017/papers/19-sortmergejoins/graefe-tkde1994.pdf>

## OS, Systems Software:

Reflections on trusting trust. Trojans in programs: <https://www.archive.ece.cmu.edu/~ganger/712.fall02/papers/p761-thompson.pdf>

Yacc (Yet another compiler-compiler) is a tool that generates a parser, the part of compiler that tries to make syntactic sense of the source code, based on a grammar provided: <http://dinosaur.compilertools.net/yacc/>

Case for a single chip multiprocessor: <http://www.cs.cmu.edu/afs/cs/academic/class/15740-f15/www/papers/onhwc96.pdf>

This paper describes the concept of RAID. Discussions here apply to the general case of reliability and fault tolerance in computer systems: <http://inst.eecs.berkeley.edu/~n252/sp07/Papers/RAID-patterson.pdf>

Microkernel OS architecture and the Mach: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.80.3969&rep=rep1&type=pdf>

Intel Pentium 4 architecture: <http://www.ecs.umass.edu/ece/koren/ece568/papers/Pentium4.pdf>.

Xen and the art of Virtualization: <https://www.cl.cam.ac.uk/research/srg/netos/papers/2003-xensosp.pdf>

Mobile Computing, the next decade: <https://www.cs.cmu.edu/~satya/docdir/satya-mobicloud-2010.pdf>

## Networking

Original ethernet paper: <http://ethernethistory.typepad.com/papers/EthernetPaper.pdf>

New directions in Cryptography: <https://ee.stanford.edu/~hellman/publications/24.pdf>

Overview of Public Key Cryptography: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.127.2652&rep=rep1&type=pdf>

Development of DNS: <http://www.cs.cornell.edu/courses/cs615/2002fa/615/mockapetris.pdf>

Tiny Tera, a packet switched core: <http://www.cs.yale.edu/homes/yry/readings/general/tinytera.pdf>

## Artificial Intelligence

The Turing Test: <https://www.csee.umbc.edu/courses/471/papers/turing.pdf>

The semantic web, a vision of the web of the future. Download the full text pdf from: <https://www.researchgate.net/publication/307845029_Tim_Berners-Lee's_Semantic_Web>

Top ten algorithms in Data Mining: <http://www.cs.umd.edu/~samir/498/10Algorithms-08.pdf>.

## Software Engineering

Essence and accident in software engineering: <http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/no-silver-bullet.pdf>

Who needs an architect: <http://files.catwell.info/misc/mirror/2003-martin-fowler-who-needs-an-architect.pdf>

Program development by stepwise refinement: <https://oberoncore.ru/_media/library/wirth_program_development_by_stepwise_refinement2.pdf>

## Storage, Databases, Distributed Systems

Scalable tolerant systems, introducing the CAP theorem: <https://pdfs.semanticscholar.org/5015/8bc1a8a67295ab7bce0550886a9859000dc2.pdf>

This paper by Google founders on how the search engine worked initially: <http://infolab.stanford.edu/~backrub/google.html>

The Pagerank algorithm used in Google search engine: <http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf>

The Google File System: <https://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf>

Mapreduce, simplified data processing on clusters: <https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf>

Google Bigtable: <https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>

Amazon’s Dynamo, the massively scalable key value store: <https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>

High-performance complex event processing over streams: <https://www2.cs.duke.edu/courses/spring08/cps399.28/papers/sigmod06-DiaoEtAl-event_processing_over_stream.pdf>

The end of an architectural era, talks about the performance issue in mainstream RDBMS systems, and that it is time for revamping their code: <http://nms.csail.mit.edu/~stavros/pubs/hstore.pdf>.

## General

Blockchain and Bitcoin. Overview: <https://www.computerworld.com/article/3191077/security/what-is-blockchain-the-most-disruptive-tech-in-decades.html>, and the first Bitcoin paper: <https://bitcoin.org/bitcoin.pdf>

# Soft Skills, Life Skills

Duration: 10 days.

Most agree that technical knowledge is just one aspect of professional life. Non-technical aspects, are equally important. Some might want to categorize it further into into non-technical skills for professional, and personal life. But considering that different people have more than just these two facets, and the fact that events in one social-professional circle affect the person, which in-turn affects what he does in other circles, we felt it was optimal to consider these together. Finally, it is people dealing with people.

## Personal Improvement

* This is the key. Not straightforward, but extremely important. Knowing what you want, doing what you would really like to do, your passion. Steve Jobs’ Stanford lecture, <https://news.stanford.edu/2005/06/14/jobs-061505/>. Followed by <https://www.nateliason.com/blog/life-you-want>, <https://www.youtube.com/watch?v=w3vgN_cqrIA>, and <https://www.telegraph.co.uk/finance/jobs/11499695/Eight-ways-to-find-the-true-passion-in-life-that-has-eluded-you.html>.
* Physical fitness: <https://www.wikihow.life/Keep-Fit>.
* Sun Salutations, a short-cut to physical and mental health: <https://www.artofliving.org/in-en/yoga/yoga-poses/sun-salutation>.
* Maintaining mental health: <https://www.mentalhealth.org.uk/publications/how-to-mental-health>.
* Relationships, Interpersonal skills: <https://www.wikihow.com/Develop-Interpersonal-Skills>, <https://www.wikihow.com/Build-Healthy-Interpersonal-Relationships>.
* Communication skills: <https://www.thebalancecareers.com/communication-skills-list-2063779>
* Balancing Perfection and 80-20 Rule: <https://personalexcellence.co/blog/80-20/>, Perfectionism vs Excellence - <http://theviewinside.me/perfectionism-vs-excellence/>
* Time Management: <https://www.skillsyouneed.com/ps/time-management.html>.
* Getting better in working better under pressure: <https://www.forbes.com/sites/forbescoachescouncil/2017/07/28/18-ways-to-get-better-at-working-under-pressure/#5101c0e32e35>.
* Spirit: Maintain spiritual health: <https://www.wikihow.com/Maintain-Good-Spiritual-Health>, and [https://www.stylecraze.com/articles/simple-steps-to-practice-spiritual-meditation/#gre](https://www.stylecraze.com/articles/simple-steps-to-practice-spiritual-meditation/#gref).

## Wealth, Investing

* Money, Currency: <https://economicsandliberty.wordpress.com/what-is-money/>, and the cryptocurrency, bitcoin: <https://economicsandliberty.wordpress.com/2011/06/22/bitcoin-a-new-commodity-created-to-serve-market-demand/>.
* Way to wealth by Benjamin Franklin: <https://www.gutenberg.org/files/43855/43855-h/43855-h.htm>.
* Investing basics: <http://www.assetmanagement.hsbc.com/in/mutual-funds/learning-centre/investment-basic/fundamentals.html>, <https://www.ways2wealth.com/investing/PostId/234/7-best-investment-solutions-for-your-capital>.
* Stock investing: Basics: <https://www.investopedia.com/university/beginner/>, Superinvestors of Graham and Doddsville, by Warren Buffet: <http://csinvesting.org/wp-content/uploads/2014/10/The-Superinvestors-of-Graham-and-Doddsville-by-Warren-Buffett.pdf>, Buffet’s advice: <https://www.businessinsider.com/warren-buffett-best-investing-advice-for-beginners-2017-11?IR=T>.

## Management and Leadership

* Principles of economics: <https://www.investopedia.com/university/economics/>.
* Principles of management: <https://2012books.lardbucket.org/books/management-principles-v1.0/s05-introduction-to-principles-of-.html>.
* Accounting: <https://www.investopedia.com/university/accounting/>.
* Business at the bottom of the Pyramid: <https://www.strategy-business.com/article/11518?gko=9a4ba>.
* Sales and Marketing: Marketing basics <https://www.business.qld.gov.au/running-business/marketing-sales/marketing-promotion/marketing-basics>, Five deadly sins in business: <https://www.processexcellencenetwork.com/lean-six-sigma-business-performance/columns/drucker-s-five-deadly-sins-in-business>, Sales <https://www.business.qld.gov.au/running-business/marketing-sales/sales/basics>.
* Writing good business proposals: <https://www.wikihow.com/Write-a-Business-Proposal>.

## Entrepreneurship

* Mouse Merchant, a Jataka Tale: <https://www.gutenberg.org/files/51880/51880-h/51880-h.htm#Page_168>.
* Starting a startup: <http://www.paulgraham.com/start.html>, and how not to start a startup: <https://medium.com/the-mission/how-not-to-start-a-startup-e87c5b344909>.
* Starting, business plans: <https://blog.hubspot.com/sales/how-to-start-a-business>.
* Management and leadership: <http://www.dii.uchile.cl/~eolguin/G%20del%20C/docs/Liderazgo/What%20leaders%20really%20do%20-%20J%20kotter.pdf>.
* How leaders accomplish more by doing less: <https://www.whatsbestnext.com/wp-content/uploads/2016/07/How-Leaders-Accomplish-More-by-Doing-Less-Whats-Best-Next.pdf>.
* Market Research: <https://www.wikihow.com/Conduct-Market-Research>, <https://blog.hubspot.com/marketing/market-research-buyers-journey-guide>.
* Presentation skills: <https://www.forbes.com/2010/02/24/effective-presentation-skills-leadership-careers-rosenthal.html#3cda54ce30cf>, Present like Steve Jobs: <https://www.youtube.com/watch?v=RHX-xnP_G5s>.
* Design Thinking: What is design thinking - <https://www.youtube.com/watch?v=pXtN4y3O35M>, How it works - <https://www.youtube.com/watch?v=U499U4TcyY8>, Is a Method not magic - <https://www.youtube.com/watch?v=vSuK2C89yjA>. Design Thinking Workshop - <https://www.youtube.com/watch?v=Z4gAugRGpeY>.
* Social Entrepreneurship using Open Source: <https://www.youtube.com/watch?v=tl58jYqCMmI>.
* Entrepreneurship case studies: <https://mitsloan.mit.edu/LearningEdge/entrepreneurship/Pages/default.aspx>.

## Social-Economic-Political Organization Schools of thought

Over centuries, many great minds have tried to visualize the ideal social-political-economic organization of society, for eventual perfect world. The struggle still continues. Were these people truly great, naive, or plain foolish? Nevertheless, here are some of them.

* Essence of Plato’s republic, aristrocracy to democracies to tyrannys: <http://factmyth.com/books/platos-republic-explained/>.
* Democracy today: <https://www.economist.com/news/essays/21596796-democracy-was-most-successful-political-idea-20th-century-why-has-it-run-trouble-and-what-can-be-do>
* Economics of Adam Smith-Marx-Keynes: <https://www.khanacademy.org/partner-content/big-history-project/acceleration/changing-economies/a/smith-marx-and-keynes>
* Cooperatives: <http://sfp.ucdavis.edu/cooperatives/what_is/>. Successes: <https://www.ijser.org/researchpaper/A-CASE-STUDY-OF-AMUL-COOPERATIVE-IN-INDIA-IN-RELATION-TO-ORGANIZATIONAL-DESIGN-AND-OPERATIONAL-EFFICIENCY.pdf>, Failures: <http://www.yourarticlelibrary.com/politics/failure-of-the-cooperative-movement-in-india/4824>.
* Liberty: <https://www.youtube.com/watch?v=8z1buym2xUM>, <https://economicsandliberty.wordpress.com/my-philosophy-of-liberty/>, <https://economicsandliberty.wordpress.com/a-way-to-be-free/>

# Project

Duration: 60 days.

In the initial days of your course itself, try to define a project for yourself based on your inclination, learnings from some business and entrepreneurship articles above, or with discussion from your seniors and mentors. A bunch of project ideas is also provided at the end of this section. You could use those.

Use learnings in this course to build up various pieces of your project. Make use of all aspects learned, right from definition, design, planning, development, testing, right up to deployment. Also sincerely follow Industry practices described in the earlier sections.

Try to put it out on github for others to see and use, and as an advertise for yourself. Try to do it in a group of 2 or 3, depending upon project size and skills needed. You could also add features to existing open source projects. See: <https://opensource.guide/how-to-contribute/>.

## Project ideas:

<https://www.quora.com/What-are-some-new-and-innovative-projects-for-a-computer-science-engineering-student>

<https://yourstory.com/2016/10/15-ideas-change-india-accenture/>

<http://bigideascontest.org/contests/all/>

<https://www.wethinq.com/en/blog/2014/02/18/32-Inspiring-Examples-of-Social-Innovation.html>

Many more mini-projects:

<https://www.geeksforgeeks.org/computer-science-projects/>

<http://nevonprojects.com/year-projects-for-computer-engineering/>

<http://services.lovelycoding.org/computer-science-project-ideas-final-year/>

<https://www.crazyengineers.com/threads/creative-crazy-ideas-for-computer-science-final-year-project.47078/>

<https://www.elprocus.com/computer-science-projects-engineering-students/>

# Verify Success

Duration: 5 days

Confirm for yourself whether you have indeed finished this course in true spirit. Use the following parameters and techniques as guiding principles:

* Whether you have earnestly attempted exercises given at the end of different sections above.
* Talk to experienced software professional(s) from time to time and get some sense of where you stand by talking to them, getting interviewed by them, etc.
* Demo your projects to experienced folks and ask them to evaluate those across various parameters. Say, put it out on Github, talk about it on blogs.
* Get formally certified by people or institutes with reasonable rigor in their evaluation process.
* Overall confirmation of your understanding (certification/diploma) is one part, but even while you are in the process of learning, keep doing regular evaluations to confirm that learning is going per expectation.

# Before Concluding

Needless to say, what we went through here is just the tip of the iceberg. Gaining expertise in any field is a long process, takes years. At the same time, there is no end to what you can learn, and everything around seems so interesting! The line must be drawn somewhere.

One smart thing to do is, maintaining general awareness of where the technology world is heading, awareness of various areas and possibilities at a high level, and developing the knack to spot upcoming learning needs. Then plan learning activity at the correct breadth and depth, identify most appropriate learning material, and do the deep dive into the area so narrowed down.

## More Topics in Computing

This course covers some important computing topics considering their applicability to all other areas. However, many other areas such as systems programming, compiler construction, graphics, and others, are important, interesting, and also need specialized training. But then, the amount of people needed to work in these areas is also limited. These people are ones who develop ‘infrastructure’ for others. One should keep building these skills over time too, but use judgement based on current area work, and accordingly prioritize.

A list of various areas, and reference to learning material follows. Even if the reference material provided here is not exactly in tune with what you want, there is nothing to worry. By now, you have developed the sense of finding appropriate learning material for yourself, from the internet, or reference books available in plenty. You just need to search smartly and select judiciously.

* **Mathematical foundations** for Computing, specifically areas such as Machine Learning, Digital Signal Processing, Computer Graphics, Networking: See section on Mathematical foundations, in the References section.
* Theory of Computation: See courses in references section.
* Algorithms: See courses in references section.
  + Advanced Algorithms
  + Graph Theory
  + Computational Geometry: <https://nptel.ac.in/courses/106102011/>
* Computer Organization: See courses in references section.
* Operating Systems: See courses in references section.
* Databases: See courses in references section.
  + Database Design
  + Database Storage
* Communications:
  + Data Networks: See courses in references section.
  + Wireless Networks: <http://nptel.ac.in/courses/106106167/>
* Cryptography and Security: See courses in references section.
* Artificial Intelligence: See courses in references section.
  + Knowledge representation and reasoning: <https://nptel.ac.in/courses/106106140/>
  + Machine Learning
  + Deep Learning (Neural nets)
  + Text Mining
  + Data Science
  + Robotics
* Information Retrieval and Web Search: See courses in references section.
* Computer Graphics: See courses in references section.
* Optimization:
  + Linear Programming: <http://nptel.ac.in/courses/112106134/>
  + Convex Optimization: See courses in references section.
* Programming: See courses in references section.
  + Object Oriented
  + Functional Programming
  + Scripting
  + Network Programming
  + Real-time programming
  + Concurrent and Parallel Programming
  + Web Programming
  + Mobile Application development: Android, IOS
  + GPU Programming
  + Language Design
  + Paradigms
* Compiler construction: See courses in references section.
* Distributed Systems: See courses in references section.
* Cloud Computing: See courses in references section.
* Systems Design and Analysis, Software Engineering: See courses in references section.
* Animation and Game Development: See courses in references section.
* Scientific, Engineering Domains: See courses in references section.
  + Digital Signal Processing:
  + Image Processing: <https://nptel.ac.in/courses/106105032/>
  + GIS and Spatial analysis
  + Clinical Data Analysis
  + Computer Vision
  + Electronic design automation: <https://nptel.ac.in/courses/106105083/>
  + CAD for VLSI:

## References

References below has a lot of pointers to great learning references to help you progress in your career on a continuous basis. Take a look at those to take stock of what all is out there with respect to what you know and don’t.

* **Rich reference of Computer Science Academic Courses**: This page lists high-quality course material of top Universities of the world in the form of assignments, lectures, notes, readings & examinations. All available online for free. See: <https://github.com/prakhar1989/awesome-courses>.
* **Academic Books**: A list of topic-wise reference books for various computer science topics is available here: <https://www.sanfoundry.com/best-reference-books-computer-science-engineering/>.
* Hacking a google interview: <http://courses.csail.mit.edu/iap/interview/materials.php>.
* For improving overall development skills, rigor, book, The Pragmatic Programmer: <https://www.nceclusters.no/globalassets/filer/nce/diverse/the-pragmatic-programmer.pdf>.
* **Mathematical foundations** for Computing, specifically areas such as Machine Learning, Digital Signal Processing, Computer Graphics, Networking.
  + See courses in references section.
  + Single variable calculus: <https://ocw.mit.edu/courses/mathematics/18-01sc-single-variable-calculus-fall-2010/>
  + Multi-variable calculus: <https://ocw.mit.edu/courses/mathematics/18-02-multivariable-calculus-fall-2007/>
  + Linear Algebra (used in Machine Learning): <https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/>
  + Probability in computing: <https://nptel.ac.in/courses/106106176/>
  + Numerical Analysis: <http://nptel.ac.in/courses/111101003/>
  + Mathematical fundamentals for machine learning: <https://www.edx.org/course/essential-math-machine-learning-python>
* More to come ...

## Payback

A lot of excellent free course material referred to above is the result of hard work put in by creators of those courses. If you feel you have benefitted, then seriously consider donating to those noble initiatives.

* MIT OCW (Open Courseware): <https://giving.mit.edu/give/to/ocw/>
* Khan Academy: <https://www.khanacademy.org/donate>
* SQLBolt: <https://sqlbolt.com/lesson/end>

# More References:

* Paper: Communicating Sequential Processes by Hoare, 1978.
* Hands-on OS internals: <https://www.ops-class.org/>.
* Lessons learned in software dev: <https://henrikwarne.com/2015/04/16/lessons-learned-in-software-development/>
* Get a practical exposure to various types of testing techniques such as unit and integration testing using python from: <https://katyhuff.github.io/python-testing/>
* Understand performance testing through: <https://software.intel.com/en-us/articles/performance-testing-and-tuning-part-1> and <https://software.intel.com/en-us/articles/performance-testing-and-tuning-part-ii>
* Understand UI testing with Python and UI testing tool Selenium: <https://www.techbeamers.com/selenium-webdriver-python-tutorial/> and then <https://www.techbeamers.com/selenium-python-test-suite-unittest/>
* Javascript build tools: <https://medium.com/the-node-js-collection/modern-javascript-explained-for-dinosaurs-f695e9747b70>
* Image Processing using Python: <https://pythonprogramming.net/loading-images-python-opencv-tutorial/>
* Hardware Control and Robotics with Python: <https://pythonprogramming.net/robotics-tutorials/>
* Consistent hashing: <https://www.toptal.com/big-data/consistent-hashing>
* Blockchain liberating potential: <https://hackernoon.com/why-everyone-missed-the-most-mind-blowing-feature-of-cryptocurrency-860c3f25f1fb>
* Program optimization: <https://en.wikipedia.org/wiki/Program_optimization>. It has some very good external links too.
* Future Outlook on Digitalization: <https://www.hhs.se/contentassets/a3083bb76c384052b3f3f4c82236e38f/managing-digital-transformation-chapter-15.pdf>
* Technical Writing: <https://blog.usejournal.com/technical-writing-basics-all-in-one-place-21cc3e477051>