Checklist for CS Degree

Keanu Daniel

24.02.2022

Contents

Introduction

Introduction to Computer Science

Topics covered: `computation`, `imperative programming`, `basic data structures and algorithms`

- ☒ Python for Everybody | 10 weeks | 10 hours/week | none | chat
- ☐ Introduction to Computer Science and Programming using Python (alt)

Core CS

Core Programming

Topics covered: `functional Programming`, `design for testing`, `program requirements`, `common design patterns`, `unit testing`, `object-oriented design`, `static typing`, `dynamic typing`, `ML-family languages (via Standard ML)`, `Lisp-family languages (via Racket)`, `Ruby`

- ☐ How to Code - Simple Data | 7 weeks | 8-10 hours/week | none | chat
- ☐ How to Code - Complex Data | 6 weeks | 8-10 hours/week | How to Code: Simple Data | chat
- ☐ Programming Languages, Part A | 5 weeks | 4-8 hours/week | How to Code (Hear instructor) | chat
- ☐ Programming Languages, Part B | 3 weeks | 4-8 hours/week | Programming Languages, Part A | chat
- ☐ Programming Languages, Part C | 3 weeks | 4-8 hours/week | Programming Languages, Part B | chat
- ☐ Object-Oriented Design | 4 weeks | 4 hours/week | Basic Java
- ☐ Design Patterns | 4 weeks | 4 hours/week | Object-Oriented Design
- ☐ Software Architecture | 4 weeks | 2-5 hours/week | Design Patterns

Core Math

Topics covered: `discrete mathematics`, `mathematical proofs`, `basic statistics`, `O-notation`, `discrete probability`

- ☐ Calculus 1A: Differentiation (alt) | 13 weeks | 6-10 hours/week | The alternate covers this and the following 2 courses | high school math | chat
- ☐ Calculus 1B: Integration | 13 weeks | 5-10 hours/week | - | Calculus 1A | chat
- ☐ Calculus 1C: Coordinate Systems & Infinite Series | 6 weeks | 5-10 hours/week | - | Calculus 1B | chat
- ☐ Mathematics for Computer Science (alt) | 13 weeks | 5 hours/week | An alternate version with solutions to the problem sets is here. Students struggling can consider the Discrete Mathematics Specialization first. It is more interactive but less comprehensive, and costs money to unlock full interactivity. | Calculus 1C | chat

CS Tools

Topics covered: `terminals and shell scripting`, `vim`, `command line environments`, `version control`

- ☐ The Missing Semester of Your CS Education | 2 weeks | 12 hours/week | - | chat

Core systems

Topics covered: `procedural programming`, `manual memory management`, `boolean algebra`, `gate logic`, `memory`, `computer architecture`, `assembly`, `machine language`, `virtual machines`, `high-level languages`, `compilers`, `operating systems`, `network protocols`

- ☐ Build a Modern Computer from First Principles: From Nand to Tetris (alt) | 6 weeks | 7-13 hours/week | - | C-like programming language | chat
- ☐ Build a Modern Computer from First Principles: Nand to Tetris Part II | 6 weeks | 12-18 hours/week | - | one of these programming languages, From Nand to Tetris Part I | chat

- ☐ Operating Systems: Three Easy Pieces | 10-12 weeks | 6-10 hours/week | - | algorithms, familiarity with C is useful | chat
- ☐ Computer Networking: a Top-Down Approach| 8 weeks | 4–12 hours/week | Wireshark Labs | algebra, probability, basic CS | chat

Core theory

Topics covered: `divide and conquer`, `sorting and searching`, `randomized algorithms`, `graph search`, `shortest paths`, `data structures`, `greedy algorithms`, `minimum spanning trees`, `dynamic programming`, `NP-completeness`

- ☐ Divide and Conquer, Sorting and Searching, and Randomized Algorithms | 4 weeks | 4-8 hours/week | any programming language, Mathematics for Computer Science | chat
- ☐ Graph Search, Shortest Paths, and Data Structures | 4 weeks | 4-8 hours/week | Divide and Conquer, Sorting and Searching, and Randomized Algorithms | chat
- ☐ Greedy Algorithms, Minimum Spanning Trees, and Dynamic Programming | 4 weeks | 4-8 hours/week | Graph Search, Shortest Paths, and Data Structures | chat
- ☐ Shortest Paths Revisited, NP-Complete Problems and What To Do About Them | 4 weeks | 4-8 hours/week | Greedy Algorithms, Minimum Spanning Trees, and Dynamic Programming | chat

Core Security

Topics covered `Confidentiality, Integrity, Availability`, `Secure Design`, `Defensive Programming`, `Threats and Attacks`, `Network Security`, `Cryptography`

Note: ***These courses are provisionally recommended***. There is an open Request For Comment on security course selection. Contributors are encouraged to compare the various courses in the RFC and offer feedback.

- ☐ Information Security: Context and Introduction | 5 weeks | 3 hours/week | - | chat
- ☐ Principles of Secure Coding| 4 weeks | 4 hours/week | - | chat
- ☐ Identifying Security Vulnerabilities | 4 weeks | 4 hours/week | - | chat

Choose **one** of the following:

- ☐ Identifying Security Vulnerabilities in C/C++Programming | 4 weeks | 5 hours/week | - | chat
- ☐ Exploiting and Securing Vulnerabilities in Java Applications | 4 weeks | 5 hours/week | - | chat

Core applications

Topics covered: `Agile methodology`, `REST`, `software specifications`, `refactoring`, `relational databases`, `transaction processing`, `data modeling`, `neural networks`, `supervised learning`, `unsupervised learning`, `OpenGL`, `raytracing`

- ☐ Databases: Modeling and Theory| 2 weeks | 10 hours/week | core programming | chat
- ☐ Databases: Relational Databases and SQL| 2 weeks | 10 hours/week | core programming | chat
- ☐ Databases: Semistructured Data| 2 weeks | 10 hours/week | core programming | chat
- ☐ Machine Learning| 11 weeks | 4-6 hours/week | linear algebra | chat
- ☐ Computer Graphics| 6 weeks | 12 hours/week | C++ or Java, linear algebra | chat
- ☐ Software Engineering: Introduction | 6 weeks | 8-10 hours/week | Core Programming, and a sizable project | chat

Core Ethics

Topics covered: `Social Context`, `Analytical Tools`, `Professional Ethics`, `Intellectual Property`, `Privacy and Civil Liberties`

☐ Ethics, Technology and Engineering| 9 weeks | 2 hours/week | none | chat
☐ Intellectual Property Law in Digital Age| 4 weeks | 2 hours/week | none | chat
☐ Data Privacy Fundamentals| 3 weeks | 3 hours/week | none | chat

Advanced CS

Advanced programming

Topics covered: `debugging theory and practice`, `goal-oriented programming`, `parallel computing`, `object-oriented analysis and design`, `UML`, `large-scale software architecture and design`

☐ Parallel Programming| 4 weeks | 6-8 hours/week | Scala programming
☐ Compilers | 9 weeks | 6-8 hours/week | none
☐ Introduction to Haskell| 14 weeks | - | -
☐ Learn Prolog Now! (alt)*| 12 weeks | - | -
☐ Software Debugging| 8 weeks | 6 hours/week | Python, object-oriented programming
☐ Software Testing | 4 weeks | 6 hours/week | Python, programming experience

Advanced systems

Topics covered: `digital signaling`, `combinational logic`, `CMOS technologies`, `sequential logic`, `finite state machines`, `processor instruction sets`, `caches`, `pipelining`, `virtualization`, `parallel processing`, `virtual memory`, `synchronization primitives`, `system call interface`

☐ Computation Structures 1: Digital Circuits | 10 weeks | 6 hours/week | Nand2Tetris II
☐ Computation Structures 2: Computer Architecture | 10 weeks | 6 hours/week | Computation Structures 1
☐ Computation Structures 3: Computer Organization | 10 weeks | 6 hours/week | Computation Structures 2

Advanced theory

Topics covered: `formal languages`, `Turing machines`, `computability`, `event-driven concurrency`, `automata`, `distributed shared memory`, `consensus algorithms`, `state machine replication`, `computational geometry theory`, `propositional logic`, `relational logic`, `Herbrand logic`, `game trees`

☐ Theory of Computation (Lectures) | 8 weeks | 10 hours/week | discrete mathematics, logic, algorithms
☐ Computational Geometry | 16 weeks | 8 hours/week | algorithms, C++
☐ Game Theory | 8 weeks | 3 hours/week | mathematical thinking, probability, calculus

Advanced math

☐ Essence of Linear Algebra | - | - | high school math | chat
☐ Linear Algebra | 14 weeks | 12 hours/week | corequisite: Essence of Linear Algebra | chat
☐ Introduction to Numerical Analysis(alt)| 7 weeks | 3-4 hours/week | Mathematics for Computer Science, Optional: Linear Algebra | chat
☐ Introduction to Logic | 10 weeks | 4-8 hours/week | set theory | chat
☐ Probability | 24 weeks | 12 hours/week | Differentiation and Integration | chat

Final project

OSS University is project-focused. The assignments and exams for each course are to prepare you to use your knowledge to solve real-world problems.

After you've gotten through all of Core CS and the parts of Advanced CS relevant to you, you should think about a problem that you can solve using the knowledge you've acquired. Not only does real project work look great on a resume, but the project will also validate and consolidate your knowledge. You can create something entirely new, or you can find an existing project that needs help via websites like CodeTriage or First Timers Only.

Students who would like more guidance in creating a project may choose to use a series of project oriented courses. Here is a sample of options (many more are available, at this point you should be capable of identifying a series that is interesting and relevant to you):

Courses	Duration	Effort	Prerequisites
Fullstack Open	12 weeks	6 hours/week	programming
Modern Robotics (Specialization)	26 weeks	2-5 hours/week	freshman-level physics, linear algebra, calculus, linear ordinary differential equations
Data Mining (Specialization)	30 weeks	2-5 hours/week	machine learning
Big Data (Specialization)	30 weeks	3-5 hours/week	none
Internet of Things (Specialization)	30 weeks	1-5 hours/week	strong programming
Cloud Computing (Specialization)	30 weeks	2-6 hours/week	C++ programming
Data Science (Specialization)	43 weeks	1-6 hours/week	none
Functional Programming in Scala (Specialization)	29 weeks	4-5 hours/week	One year programming experience
Game Design and Development with Unity 2020 (Specialization)	6 months	5 hours/week	programming, interactive design

Evaluation

Upon completing your final project: - Submit your project's information to PROJECTS via a pull request. - Put the OSSU-CS badge in the README of your repository!

- Use our community channels to announce it to your fellow students.

Solicit feedback from your OSSU peers. You will not be "graded" in the traditional sense — everyone has their own measurements for what they consider a success. The purpose of the evaluation is to act as your first announcement to the world that you are a computer scientist and to get experience listening to feedback — both positive and negative.

The final project evaluation has a second purpose: to evaluate whether OSSU, through its community and curriculum, is successful in its mission to guide independent learners in obtaining a world-class computer science education.