

Core Code Concepts

By Jeremy Griffith

# Document Purpose

The purpose of this document is to provide a basic Computer Science study guide to people who are taking introductory programming courses, studying for basic coding tests, or planning to obtain a software certification. You may be interested in this study guide if you are:

* Taking a Computer Science AP Course
* Studying for the Computer Science AP Test
* Registering for an Introductory Programming College Course
* Brushing up on Concepts for the Major Field Test in Computer Science
* Refreshing Core Skills for a Computer Science Certification Exam

## What this Guide Will Do

This guide will provide you with an overview of all the core competencies needed to be a well-rounded Computer Scientist. The following is a breakdown of the entire guide:

1. [The Core Concept Checklist](#_The_Core_Concept)
2. [Study Plan](#_The_Study_Plan)s
3. [Topic Breakdowns](#_Topic_Breakdowns)
4. [Nomenclature](#_Nomenclature)

The purpose of the Core Concept Checklist is to give you a way to keep track of all the concepts you’ve studied. Then in the Study Plans section, you will find different study packs to suit your needs. For further clarification each topic can be found in the Topic Breakdown section. In order to keep the Topic Breakdown section light, definitions have been placed in a separate Nomenclature section.

## What this Guide Will **NOT** Do

This guide will not go into a deep dive of any of the subjects described. Likewise, none of topics will be covered using a specific language. Topics may reference a language, but this guide is purely conceptual and theoretical.

That said, I wouldn’t want to leave you hanging. That’s why I have a [website](https://therenegadecoder.com/) loaded with topics that go much further in depth than this document. If you visit, you should be able to find an explanation of just about every topic described in this document. If not, I’ve listed a few of my favorite go-to resources below:

* [CodeAcademy](https://www.codecademy.com/)
* [Computerphile](https://www.youtube.com/user/Computerphile)
* [Khan Academy](https://www.khanacademy.org/)
* [Learn with Unity](https://unity3d.com/learn/)
* [MIT OpenCourseWare](http://ocw.mit.edu/courses/#electrical-engineering-and-computer-science)
* [Quora](https://www.quora.com/)
* [StackOverflow](https://stackoverflow.com/)

# The Core Concept Checklist

The purpose of this section is to provide you with a way of measuring your progress as you study through the material. As you get comfortable with a particular topic, you can mark it as complete here.

* [Algorithms](#_Algorithms)
* Artificial Intelligence
* Computer Architecture
* Computer Graphics
* Computer Networks
* Computer Security
* Database Systems
* [Data Structures](#_Data_Structures)
* Digital Systems Design
* Discrete Math
* Distributed Computing
* Embedded Systems
* Game Development
* Human-Computer Interaction
* Modeling & Simulation
* [Object Oriented Programming](#_Object_Oriented_Programming)
* Operating Systems
* [Programming Languages](#_Programming_Languages)
* Robotics
* Software Engineering

# Study Plans

The purpose of this section is to provide you with a few suggested study plans for those that are looking for a bit more guidance. The study plans are separated by category to ensure maximum learning efficiency. The following is a list of categories:

* The Application Pack
  + Game Development
  + Human-Computer Interaction
  + Modeling and Simulation
  + Robotics
  + Software Engineering
* The Hardware Pack
  + Computer Architecture
  + Computer Graphics
  + Digital Systems Design
  + Embedded Systems
* The Information Technology Pack
  + Computer Networks
  + Computer Security
  + Database Systems
  + Distributed Computing
  + Operating Systems
* The Theory Pack
  + Algorithms
  + Artificial Intelligence
  + Data Structures
  + Discrete Math
  + Object Oriented Programming
  + Programming Languages
* The Total Pack
  + All Subjects

Each study plan is designed for a 4-week study period. Each week is designed to cover 3 major topics. These plans assume that you already have some basic knowledge in the subject areas. If not, it may be advantageous to stretch the study plan to your needs.

## The Application Pack

The Application Pack focuses on real-world applications of Computer Science concepts. In this pack, the following concepts are covered:

* Game Development
* Human-Computer Interaction
* Modeling and Simulation
* Robotics
* Software Engineering

|  |  |  |  |
| --- | --- | --- | --- |
| Application Plan | Topic 1 | Topic 2 | Topic 3 |
| Week 1 |  |  |  |
| Week 2 |  |  |  |
| Week 3 |  |  |  |
| Week 4 |  |  |  |

## The Hardware Pack

The Hardware Pack focuses on Computer Science concepts as they apply to digital and hardware design. In this pack, the following concepts are covered:

* Computer Architecture
* Computer Graphics
* Digital Systems Design
* Embedded Systems

|  |  |  |  |
| --- | --- | --- | --- |
| Hardware Plan | Topic 1 | Topic 2 | Topic 3 |
| Week 1 |  |  |  |
| Week 2 |  |  |  |
| Week 3 |  |  |  |
| Week 4 |  |  |  |

## The Information Technology Pack

The Information Technology Pack focuses on Computer Science topics related to networks and services. In this pack, the following concepts are covered:

* Computer Networks
* Computer Security
* Database Systems
* Distributed Computing
* Operating Systems

|  |  |  |  |
| --- | --- | --- | --- |
| IT Plan | Topic 1 | Topic 2 | Topic 3 |
| Week 1 | Encryption | SQL |  |
| Week 2 |  |  |  |
| Week 3 |  |  |  |
| Week 4 |  |  |  |

## The Theory Pack

The Theory Pack focuses on core Computer Science topics. In this pack, the following concepts are covered:

* Algorithms
* Artificial Intelligence
* Data Structures
* Discrete Math
* Object Oriented Programming
* Programming Languages

While the theory pack is the largest pack, it’s expected that you’re already familiar with many of the concepts. If not, it may be a good idea to extend this pack over a longer period.

|  |  |  |  |
| --- | --- | --- | --- |
| Theory Pack Plan | Topic 1 | Topic 2 | Topic 3 |
| Week 1 | Boolean Algebra | Automata Theory | Big O Notation |
| Week 2 | Classes/Objects | Parse Trees | Lists, Trees, Graphs |
| Week 3 | Inheritance/Polymorphism | Type Checking | Design Patterns |
| Week 4 | Machine Learning | Code Generation | Algorithms |

The goal of the Theory Pack schedule is to gradually cement Computer Science concepts through the snowball effect. In the early stages of your studies, you’ll cover basic discrete math concepts that’ll be used in algorithm design and analysis down the road. By the middle of your studies, you should feel more confident coding up and using the basic data structures. Toward the end of your studies, you should have a solid foundation in theoretical Computer Science. This will be handy when working with algorithms, exploring machine learning, and writing your own compiler or interpreter.

## The Total Pack

The Total Pack is a sort of SparkNotes-style pack which focuses on breadth rather than depth. In this pack, each topic will be covered using its highlights under the assumption that you’re just looking for a refresher.

|  |  |  |  |
| --- | --- | --- | --- |
| Total Plan | Topic 1 | Topic 2 | Topic 3 |
| Week 1 | Object Oriented Programming | Discrete Math | Data Structures |
| Week 2 | Digital Systems Design | Algorithms | Software Engineering |
| Week 3 | Programming Languages | Artificial Intelligence | Operating Systems |
| Week 4 | Embedded Systems | Modeling & Simulation | Robotics |

The goal of the Total Pack schedule is to build up knowledge incrementally from week to week. That should be pretty clear as the first week covers Computer Science basics. The second week is more focused on furthering the concepts in the first week with some deeper theory. By the third week, we’re looking to hit some of the more challenging computer science concepts. During the last week, we’re just looking to tackle some more applied fields.

# Topic Breakdowns

The purpose of this section is to provide an overview of each topic with a list of subtopics to explore.

## Algorithms

Algorithms are problem solving techniques which are used by every facet of Computer Science. These techniques are characterized by the methods they use to solve generic types of problems. If you’re looking to study algorithms, a good place to start would be to get familiar with the following types of problems:

* Combinatorial Algorithms
* Computational Math & Science Algorithms
* Computer Science Algorithms
* Signal Processing Algorithms

In addition, it’s probably a good idea to really dig deep into the following famous algorithms:

* Breadth & Depth First Search Algorithms
* Data Compression Algorithms
* Dijkstra’s Algorithm
* Integer Factorization Algorithm
* Map-Reduce Algorithm
* Merge Sort & Quick Sort Algorithm
* Random Number Generation Algorithms
* RSA Algorithm

Rather than studying algorithms directly, you can study algorithm fundamentals. The following topics are great if you want to get a better understanding of general algorithm design and analysis:

* Asymptotic Notation
* Divide and Conquer
* Dynamic Programming
* Loop Invariants
* Recurrence Relations

## Artificial Intelligence

## Computer Architecture

## Computer Graphics

## Computer Networks

## Computer Security

## Database Systems

## Data Structures

Data structures is a fun area of Computer Science because it’s all about data organization. In particular, we care about storing data in way that it utilized efficiently. In general, the data structures topic is all about studying the core families through the lens of [Big O Notation](https://therenegadecoder.com/java/big-o-notation-data-structures/). The following is a list of common data structures to study:

* Arrays
* Array Lists
* Graphs
* Hash Tables (Associative Arrays)
* Linked Lists
* Stacks
* Trees
* Queues

After you have a solid grasp of the basic data structures, it’s probably a good idea to go ahead and use a few in actual application.

## Digital Systems Design

## Discrete Math

Mathematics play an important role in all of Computer Science. However, discrete math is of particular interest to use because it forms the basis of algorithm verification.

## Distributed Computing

## Embedded Systems

## Game Development

I chose to make game development its own topic because the area is just too unique and expansive to fold into some of the other topics like computer graphics or human-computer interfaces. In fact, game development is its own beast. It pulls together concepts from physics, algorithms from graphics, designs from human-computer interfaces, and much more. If you’re interested in learning more about game development, the following engines are good place to start:

* Blender
* Unity
* Unreal

Beyond engines, it’s a good idea just to get familiar with some of the other topics listed previously. They’ll aid you well in game development.

## Human-Computer Interaction

## Modeling & Simulation

## Object Oriented Programming

Object oriented programming (OOP) is a type of programming where program state is preserved and manipulated through objects. Objects allow us to create models of real world phenomenon as an effort improve code maintainability. The goal of OOP is to improve maintainability of large software systems. Whether OOP actually achieves that or not is up to the developer community to decide.

Common languages that support OOP include:

* C++
* C#
* Java
* Python
* Swift

If the goal is to be proficient in OOP, the following topics are recommended:

* Abstraction
* Classes/Objects
* Design Patterns
* Encapsulation
* Generic Types
* Inheritance
* Polymorphism
* SOLID & GRASP Principles

Studying these concepts is generally not enough to be proficient in the subject. In addition, you should probably put OOP into practice in one way or another. A good starting point is to try to code up some of the basic data structures by hand.

## Operating Systems

## Programming Languages

Programming languages are an area of Computer Science which defines the way we interface with computer hardware. Beyond syntax, there’s a whole host of topics that can be explored such as:

* Compilers
* Grammars
* IDEs
* Interpreters
* Middle-Level Languages
* Static Analysis
* Type Systems

These topics are heavy hitters for sure. Writing a compiler or interpreter is no laughing matter, but the process would teach you just about everything you would need to know about programming languages.

## Robotics

## Software Engineering

# Nomenclature

The purpose of this section is to provide you with a focused list of all the terms you’ll need to understand if you’re studying Computer Science or a related field.

**Algorithm**: a set of instructions for solving a problem

**Application Program Interface (API)**: a collection of exposed interfaces and protocols for the purpose of general reuse

**Concatenation**: the process of combining things together like a pair of strings or several lists

**Constant**: a value that is fixed and cannot be changed

**Comment**: an annotation of a line or section of code

**Compiler**: a software system which converts code to lower-level code

**Documentation**: literature that provides details about a library or tool

**Function**: a set of instructions that can be referenced by a name

**Graphical User Interface**: an interface which allows the use of electronic devices via graphical icons and visual cues

**Integrated Development Environment (IDE)**: a source code editor with automation tools

**Interpreter**: a software system which converts code to lower-level code on-the-fly

**Library**: a collection of implementations for the purpose of general reuse

**Loop**: a repeated sequence of instructions until some condition is met

**Method**: a function that is associated with an object

**Method Overloading**: the ability to define multiple functions in the same scope with the same name

**Method Overriding**: the ability of a subclass to create a more specific version of a method already provided by its superclass

**Multithreading**: the ability of a processor to execute multiple sets of instructions concurrently while sharing the same resources

**Polymorphism**: the ability of a variable, method, or object to exist in multiple forms

**Pseudo Code**: an informal or simplified programming language used to describe how a program should execute

**Readability**: the measure of ease of interpretation and understanding of source code

**Recursion**: a method of problem solving which derives the solution from solutions to smaller sections of the same problem

**Variable**: a value that is subject to change

**Version Control**: a system that supports organization of many versions of software