# Course Description

The art and science of computation predates the modern computer by centuries, and in fact is built upon a double foundation of formal logic (going all the way back to authors such as Aristotle) and mathematical reasoning such as set theory, algorithms, and functions. The modern computer is a physical model for repeated uses Boolean logic in the same way that the abacus is a physical model for repeated steps for performing numerical computation.

The goals of this course are to give students (1) a grounding in formal logic from Aristotle through Boole, (2) a foundation in the mathematics relevant to the theory of computation, (3) the philosophical connection between the two, (4) initiation into seeing the world through the lens of computation, which is proving a rethinking of the world on a scale at least as large as the introduction of calculus caused, and (5) an understanding of the limits of formal logic and computation, including the need for heuristics alongside algorithms because of either computational complexity or computational impossibility. All of this is oriented towards the end of getting students to think computationally and algorithmically.

Twenty-five out of a total of 36 weeks of the course are spent on pure logic and the Theory of Computation. These topics are in part demonstrated in the remaining weeks by utilizing programing concepts, principally Java. However, the use of the Java programing language is to help establish these mathematical principles.  For this reason, this course should be considered a mathematics course and not a technology course.

The logic portion includes everything necessary to understand the basic underpinnings of mathematical logic and syllogistic reasoning, including proofs of theorems about logical statements (DeMorgan’s Laws, etc.). The topics from the Theory of Computation include formal languages, regular expressions, deterministic and non-deterministic finite automata, and Turing machines. Along the way, students will employ famous mathematical algorithms, e.g. Euclid’s algorithm for finding the greatest common divisor, and various mathematical topics necessary to understand these algorithms, including modular arithmetic and base systems other than base ten.

|  |  |  |
| --- | --- | --- |
| Week 1: Logical Foundations for Mathematics and Computation |  | Quarter 1 |
| **Weekly Overview** | |  |
| This week presents an introduction to logic, stressing the distinction between formal and material logic and laying the groundwork for forthcoming computational concepts. The idea is to frame the course for what it is: a computer is a physical model of a formal logical system in the same way that an abacus is a physical model of arithmetic. In this week, the introduction of disjunctive and conjunctive is an important moment. It introduces *and* and *or*, which sets the stage for everything that is to come. | |  |

|  |  |  |
| --- | --- | --- |
| Week 2: Logical Foundations for Mathematics and Computation |  | Quarter 1 |
| **Weekly Overview** | |  |
| This week focuses on the formalization of logic in truth tables, truth trees, and to some degree sets, but also situates formal logic properly by returning to judgment and categorical propositions. Students will diagram formal propositions with *and*, *or*, and *not* using truth tables and truth trees, understand the limits of bivalence, prove formal equivalencies with truth tables, and understand the four types of categorical propositions, including distributed and undistributed terms. | |  |

|  |  |  |
| --- | --- | --- |
| Week 3: Logical Foundations and Conditional Programming |  | Quarter 1 |
| **Weekly Overview** | |  |
| This week focuses on the Square of Opposition, obversion, conversion, and contraposition, which paves the way for their mathematical equivalences of inverse, convers, and contrapositive. Now that students have a grounding in truth tables and the logical operations of *and*, *or*, and *not*, they are ready to see how these concepts play out in a formal programming language like Java. Students will explore conditional programming and connect the statements to what they have learned about truth tables. | |  |

|  |  |  |
| --- | --- | --- |
| Week 4: Logical Foundations and Conditional Programming |  | Quarter 1 |
| **Weekly Overview** | |  |
| The focus for this week is on the formal syllogism, which ties together the logical topics students have already studied by adding *implication*. This is paired with the mathematical version of conditionals and biconditionals. It includes the six rules for validity in a syllogism, and the various types of categorical syllogisms. Students will also see in a more formal way the elements of conditional programming, whereas the previous week was an opportunity to explore these concepts. As part of this instruction, students will also see data types and their operations. This includes instruction in modular arithmetic and a review of the division algorithm. | |  |

|  |  |  |
| --- | --- | --- |
| Week 5: Circuits, Logic Gates, and Iteration |  | Quarter 1 |
| **Weekly Overview** | |  |
| The focus this week takes a giant step forward in connecting formal logic with the machine of a computer. Students have now had truth tables, and they have also spent quite a bit of time learning that formal logic alone does not convey truth. They are ready to see how a “computing device” might be built. To do this, we use series and parallel circuits to model *and* and *or*, and we take the next step to logic gates. In all of this, we take our lead from W. Daniel Hillis in *The Pattern on the Stone*. Students will wire a simple “computer” that models Boolean logic using what they know about *and* and *or*. At the end of the week, students will learn about actual logic gates and redo their lab (this time on paper) using these more complicated devices. Although the study of logic will continue, this week marks a culmination of sorts as it present the computer for what it is: a physical device that models Boolean logic. | |  |

|  |  |  |
| --- | --- | --- |
| Week 6: Iteration |  | Quarter 1 |
| **Weekly Overview** | |  |
| This week focuses on the for loop as a particular incarnation of iteration, a necessary step towards a formal “algorithm.” While this is a construct from coding, the real focus this week is on important/famous mathematical algorithms: prime checking factoring numbers, and summing odd numbers to produce squares. Iterative thinking is at least some perquisite for the upcoming material on deterministic finite automata, and is certainly relevant for the much later material on Turing Machines. | |  |

|  |  |  |
| --- | --- | --- |
| Week 7: Iteration |  | Quarter 1 |
| **Weekly Overview** | |  |
| This week continues the discussion about iteration and adds the construct of the while loop within Java. The students will continue to program famous mathematical algorithms, such as finding the nth Fibonacci number, Euclid’s algorithm for finding the greatest common factor, and finding the least common multiple, and will explore some unsolved math problems such at the Collatz conjecture, the twin prime conjecture, and the question of the existence of odd perfect numbers. | |  |

|  |  |  |
| --- | --- | --- |
| Week 8: Formal Languages and Regular Expressions |  | Quarter 1 |
| **Weekly Overview** | |  |
| This week focuses on formal languages and regular expressions. This is important because it provides background for the material on Turing Machines. Students will be introduced to formal languages (including the mathematical properties of formal languages) and to regular expressions and their mathematical properties. They will go back and forth between English descriptions of a regular language and the regular expression that describes the language. They will also be introduced to deterministic and non-deterministic finite automata and Kleene’s Theorem. | |  |

|  |  |  |
| --- | --- | --- |
| Week 9: Regular Expressions Lab |  | Quarter 1 |
| **Weekly Overview** | |  |
| In this week, students will see the concept of regular expression play out in Java code. This is to reinforce the concepts, but also to extend their knowledge by seeing complicated regular expressions and their power for describing patterns. | |  |

|  |  |  |
| --- | --- | --- |
| Week 10: Methods |  | Quarter 2 |
| **Weekly Overview** | |  |
| This week focuses on methods. This is the first abstraction in computation, (second if you include variables). The mathematical presentation of this is the *function.* Students will connect method to functions (noting that in some languages, they are actually called “functions”), and learn to write and use methods. There will be a return to the regular expression lab that will make use of methods. This is also the end of the first quarter, which nicely culminates in the writing of methods and includes a continued emphasis on regular expressions. | |  |

|  |  |  |
| --- | --- | --- |
| Week 11: Arrays |  | Quarter 2 |
| **Weekly Overview** | |  |
| This week focuses on arrays, which necessarily includes an emphasis on iteration and methods. For arrays, students should know how they are stored and indexed and how they are affected when they are parameters in a method. There will be a lab that includes a very famous algorithm, the Sieve of Eratosthenes. | |  |

|  |  |  |
| --- | --- | --- |
| Week 12: Searching and Complexity |  | Quarter 2 |
| **Weekly Overview** | |  |
| This week begins a more formal presentation of algorithms by looking at searching algorithms and complexity analysis. Students should be able to code linear and binary search and should be able to conceptually define *O*(*n*) functions that describe the efficiency of an algorithm. This week presents significant mathematical concepts. The concept of *O*(*n*) requires a robust understanding of functions and limits, and the entire library of functions (linear, quadratics, polynomial, exponential, logarithmic, etc.) from the Algebra II and Calculus courses is necessary for a full understanding of complexity analysis. This week demonstrates why Calculus is a prerequisite for this course. | |  |

|  |  |  |
| --- | --- | --- |
| Week 13: Sorting and the Beginning of Recursion |  | Quarter 2 |
| **Weekly Overview** | |  |
| This week focuses on sorting algorithms, a continued exploration of complexity analysis, and an introduction to recursion. Students will program three quadratic sorts and perform complexity analysis on them. They will also being a discussion of recursion by connecting it back to recursive mathematical formulas. | |  |

|  |  |  |
| --- | --- | --- |
| Week 14: Recursion |  | Quarter 2 |
| **Weekly Overview** | |  |
| This week is a deep dive into recursion. Students will see examples when the recursive call is in the beginning, the middle, and the end. They will program several examples of algorithms they have already seen (gcd, primes, etc.), and also program Merge Sort. Complexity analysis of recursive algorithms requires students to learn how to work with turning recursive sequences into closed, explicit formulas. With Merge Sort, they see their first non-quadratic sort and learn about the behavior of the function *f*(*n*) = *n*log*n*. | |  |

|  |  |  |
| --- | --- | --- |
| Week 15: P/NP/NP Complete |  | Quarter 2 |
| **Weekly Overview** | |  |
| This week presents, in light of previous material, the first real limitation of a computer. This is a practical limitation, not a theoretical limitation. The idea is that some things *can* be coded, but their best-known algorithms are too slow to be useful. Students will learn about the categories of P, NP, and NP Complete, and understand that this is the number one unsolved problem in computer science. This looks ahead to heuristics (what do you do if you cannot write an algorithm because either (a) you don’t have the information, or (b) it will take too long). Finally, students will program an NP Complete problem to demonstrate that (1) the algorithm is not difficult, but (2) it takes way too long to run for large cases. | |  |

|  |  |  |
| --- | --- | --- |
| Week 16: Objects |  | Quarter 2 |
| **Weekly Overview** | |  |
| This week contains the second abstraction: Objects. In “methods”, students abstracted “verbs”. Here they abstract “nouns” by writing their own data types. Students will generate their first data structure: the ArrayList, which is like an array, but has no bound on size, no gaps, and can have elements inserted and deleted. This is laying the groundwork, too, for inheritance and polymorphism. The discussion of the ArrayList methods includes the mathematics of the storage as well as complexity analysis. | |  |

|  |  |  |
| --- | --- | --- |
| Week 17: Physical Computation |  | Quarter 2 |
| **Weekly Overview** | |  |
| In this week, students take a massive step forward in understanding how a computational device might actually be structured. The week begins with a study of different base systems, with an emphasis on binary arithmetic. This is connected to formal logic in an obvious way (a bivalent system, consisting only of True and False, is inherently binary). Students will review logic gates and circuits, and then study the half Adder and adder as well as flip flops and clocked systems. | |  |

|  |  |  |
| --- | --- | --- |
| Week 18: Semester Exam |  | Quarter 2 |
| **Weekly Overview** | |  |
| This week is exam week. | |  |

|  |  |  |
| --- | --- | --- |
| Week 19: Logical Foundations for Mathematics and Computation |  | Quarter 3 |
| **Weekly Overview** | |  |
| At the start of the second semester, we return back to formal logic and the syllogism. This is to serve two purposes. First, in placing the syllogism here, we give a much-needed reminder of the distinction between formal and material logic. Second, it sets the stage for a return to truth trees, which, together with polymorphism, will form the basis of the upcoming lab. | |  |

|  |  |  |
| --- | --- | --- |
| Week 20: Logical Foundations for Mathematics and Computation |  | Quarter 3 |
| **Weekly Overview** | |  |
| Continuing the work from last week, students will work with syllogisms of the first figure, diagramming them with Venn diagrams and stating their Set Theory equivalences. They will even explore hypothetical syllogisms and disjunctive and conjunctive syllogisms. By the end of these two weeks, students will have acquired the basic skills necessary for mathematical theorem proving. | |  |

|  |  |  |
| --- | --- | --- |
| Week 21: Inheritance and Polymorphism |  | Quarter 3 |
| **Weekly Overview** | |  |
| This week focuses on inheritance and polymorphism. Students will first discuss Porphyry’s *Isagoge*, which provides a beautiful parallel with the hierarchy of inheritance in code. Students will write a set of classes with inheritance, and will explore how polymorphism impacts the execution of methods at run time. They will make a distinction between compile time reference type and run time construction type. | |  |

|  |  |  |
| --- | --- | --- |
| Week 22: Polymorphism Lab |  | Quarter 3 |
| **Weekly Overview** | |  |
| This week is a robust lab experience involving inheritance and polymorphism. | |  |

|  |  |  |
| --- | --- | --- |
| Week 23: The Parse Tree Logic Lab |  | Quarter 3 |
| **Weekly Overview** | |  |
| This week is a major programming lab that brings together formal logic (truth trees), inheritance, polymorphism, data structures, and recursion. Students will need to write the various types of Nodes for a logic tree structure. The actual tree class will also required additional functionality. At a minimum, this should involve evaluate(), print(), depth(), numberOfLeaves(), numbersOfNonLeaves(), and numberOfNodes(). This lab brings together logic trees, objects, inheritance, polymorphism. While coding is involved, the bulk of the material in this week is mathematical. | |  |

|  |  |  |
| --- | --- | --- |
| Week 24: Stacks |  | Quarter 3 |
| **Weekly Overview** | |  |
| This week focuses on the Stack data structure, but also introduces the notion of Nodes. Students will have been exposed to the idea during the Tree lab, but there they will have just been given there and not asked to work with them. In writing the Stack this way, this also gives students a way to see two different implementations of the same interface – they first being the array-based Stack that they saw during the semester exam. Finally, students will use the Stack to solve a programming problem. | |  |

|  |  |  |
| --- | --- | --- |
| Week 25: Hash Tables |  | Quarter 3 |
| **Weekly Overview** | |  |
| This week focuses on the definition of a Hash Table. Topics here heavily rely on the mathematical concept of a function, as well as modular arithmetic. Students will explore the hash function, mathematical properties of a good hash function, and techniques for handling collisions and techniques for accepting collisions when they do happen. | |  |

|  |  |  |
| --- | --- | --- |
| Week 26: Heuristics |  | Quarter 3 |
| **Weekly Overview** | |  |
| As we get closer to the end of the year, we return to the theme of what a computer *cannot* do. We saw in the discussion of NP that some things are simply not efficient enough to be done. When this happens, we employ a heuristic rather than an algorithm. We also employ heuristics when we do not have all of the information. Students will learn the difference between a heuristic and an algorithm, and they will write a heuristic function for an intractable problem, emphasizing that *human judgment* provide the rationale for the decision. Students will change the heuristic and observe the differences. | |  |

|  |  |  |
| --- | --- | --- |
| Week 27: Heuristics Lab |  | Quarter 3 |
| **Weekly Overview** | |  |
| This week focuses on a major lab in which students are to write a heuristic to solve a complicated problem. | |  |

|  |  |  |
| --- | --- | --- |
| Week 28: Turing Machines, Computational Power, and von Neumann Architecture |  | Quarter 4 |
| **Weekly Overview** | |  |
| This week wraps up the progression from formal logic the working computer. Students have seen various pieces of this throughout the year, but it all comes together in the Turing Machine. Students will read from Turing directly. They will also discuss the Universal Turing Machine and writings from Babbage and Lovelace, as well as von Neumann architecture. It is setting stage for next week’s culmination on the limits of computation. | |  |

|  |  |  |
| --- | --- | --- |
| Week 29: The Limits of Computation |  | Quarter 4 |
| **Weekly Overview** | |  |
| This week culminates in the Halting Problem, which definitively sets limits on the power of computation. Students will also learn that a formal system can never decide the universal truth value of a logical statement, a powerful limitation of computation that is directly related to the Halting Problem. | |  |

|  |  |  |
| --- | --- | --- |
| Weeks 30–34: Final Project |  | Quarter 4 |
| **Weekly Overview** | |  |
| The final project is a five-week major assignment. It’s theme will vary form year to year, but will always be centered around a major mathematical topic that needs to be solve or implemented using computational techniques. The first example of this is to implement RSA encryption. This process involves advanced theorems from modular arithmetic, factoring large numbers, and (the way we have it structured), the ability to implement “famous” arithmetic algorithms for very large numbers. | |  |

|  |  |  |
| --- | --- | --- |
| Week 36: Final Exam |  | Quarter 4 |
| **Weekly Overview** | |  |
| This week is for final exams. | |  |