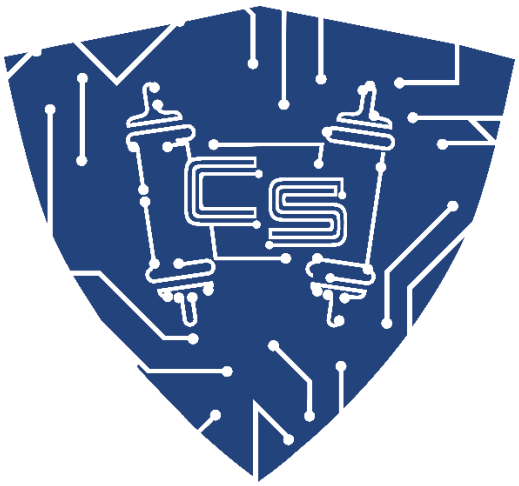
The Guide to the Computer Science Major at Yeshiva College

Revision: 3.0
(released October 2024)



Industry Advisory Board

This C.S. curriculum was developed in consultation with our industry advisory board, listed (in alphabetical order by last name) below. Their input and guidance is there to ensure that you have the opportunity to prepare for a long and successful career in C.S.

- **Dr. Henrique Andrade:** Principal Engineer, Shopify
- **Dr. Erick Brethenoux:** Chief of Research, Artificial Intelligence, Gartner; adjunct professor, Stuart School of Business at Illinois Institute of Technology
- **Steve Demuth:** CTO, Mayo Clinic (retired)
- **Rich Dutton:** Senior Staff Software Engineer, Cloud Machine Learning Scaling, Google.
- **Dr. Aliza Heching:** Research Staff Member at IBM Research
- **Dr. Jonathan Hosking:** Principal Research Scientist, Amazon
- **Paul McGregor:** Technology Fellow and Managing Director, Goldman Sachs
- **Dr. Howard Morgan:** Chairman, B Capital Group
- **Dr. Claudia Perlich:** Head of Strategic Data Science, Investment Management, TwoSigma
- **Dr. Kavitha Srinivas**: Research Staff Member, IBM
- **Dr. Joel Wein:** Senior Director of Engineering, Infrastructure Storage, Google

This guide includes:

1. Tracks in the Major and What Courses to Take Each Semester
2. Y.C. Core and Honors Thesis Requirements
3. Course Descriptions
4. Prerequisite Flow Charts
5. Yeshiva College Computer Science Faculty

For answer to frequently asked questions, please visit:
https://www.yu.edu/yeshiva-college/ug/computer-science

1. Tracks in the Major and What Courses to Take Each Semester

The C.S. department offers the following two **Bachelor of Science** tracks for C.S. majors:

- **Distributed Systems:** Learn to build the large scale software systems that are critical in almost every major industry today: Internet giants (Google, Amazon, Facebook, Netflix, etc.), Finance (trading, high speed risk modeling, etc.), Logistics (FedEx, UPS, etc.), Data-driven medicine (Mayo Clinic, Memorial Sloan Kettering, etc.), Transportation (Uber, Lyft, autonomous vehicles), etc. all build, use, and depend on distributed systems. **This track covers:**
    - Computer Science and software engineering **fundamentals**
    - In-depth applied knowledge of **every level of the software stack**: system, operating system, network, application, database, and distributed
    - Building software **at any scale** – a single process on a single computer, multiple processes or threads on a single computer, distributed across a single cluster, and distributed across the internet, a.k.a. **cloud computing**.
    - **Deep understanding** of the architecture and proper use of a range of **database technologies**. (SQL, NoSQL and NewSQL)
    - Building **secure software**

- **Artificial Intelligence:** Data-rich industries (e.g. tech, finance, marketing, logistics) are increasingly using statistical & probabilistic approaches in mission-critical decision making and software systems. **This track covers:**
    - Computer Science and software engineering **fundamentals**
    - In-depth applied knowledge of major aspects of AI: **Artificial Intelligence, Machine Learning, and Natural Language Processing.**

Below is a summary comparison of the different tracks:

	Distributed Systems	A.I.
General Software Engineering	Best	Better
Cloud Computing	Best	Good
Cybersecurity	Best	N/A
Machine Learning	N/A	Best
Artificial Intelligence, Natural Language Processing	Good	Best
Years	4	4
Courses	20	22
Credits	67	74
I should do this track if…	**I enjoy building**	**I enjoy analyzing**
Is going to graduate school necessary to get, and keep, great jobs?	**No**	**No**

Color key, from best to worst
Best
Better
Good
N/A

Note: the "building vs. analyzing" dichotomy above regarding choice of tracks should not be understand as being absolute; those working in Artificial Intelligence will build, and those working in Distributed Systems will analyze. Those descriptions are meant as a topology of personalities and inclinations, not as a strictly disjoint or binary choice.

Distributed Systems Track

(20 Courses, 67 Credits, 4 Years)

Semester-By-Semester Schedule

Year on Campus	Fall Semester	Spring Semester
1st	Intro to C.S. (COM 1300)	Data Structures (COM 1320)
	Calculus I (MAT 1412)	***Linear Algebra (MAT 2105)***
	YC Core #1	Mathematics for Computer Science (COM 1310)
		YC Core #2
2nd	Intro to Algorithms (COM 2545)	Design & Analysis of Algorithms (COM 2546)
	Computer Organization (COM 2113)	Operating Systems (COM 3610)
	YC Core #3	**YC Core #4**
3rd	Introduction to Distributed Systems (COM 3800)	Advanced Distributed Systems (COM 3810)
	Parallel Programming (COM 3820)	CyberSecurity (COM 4580)
	Networking (COM 2512)	Modern Data Management (COM 3580)
	YC Core #5	**YC Core #6**
4th	Programming Languages (COM 3640)	Compilers & Tools (COM 3645)
	Database Implementation (COM 3563)	Capstone Project (COM 4020)
	Artificial Intelligence (COM 3760)	**YC Core #8 - ELECTIVE**
	YC Core #7 - ELECTIVE	

Artificial Intelligence Track

(22 Courses, 74 Credits, 4 Years)

Semester-By-Semester Schedule

Year on Campus	Fall Semester	Spring Semester
1st	Intro to C.S. (COM 1300)	Data Structures (COM 1320)
	Calculus I (MAT 1412)	***Calculus II (MAT 1413)***
	YC Core #1	Mathematics for Computer Science (COM 1310)
		YC Core #2
2nd	Introduction to Algorithms (COM 2545)	Design & Analysis of Algorithms (COM 2546)
	Linear Algebra (MAT 2105)	***Multivariable Calculus (MAT 1510)***
	Computer Organization (COM 2113)	***Probability Theory (MAT 2461)***
	YC Core #3	**YC Core #4**
3rd	Artificial Intelligence (COM 3760)	Machine Learning (COM 3920)
	Mathematical Statistics (MAT 2462)	Modern Data Management (COM 3580)
	Programming Languages (COM 3640)	Operating Systems (COM 3610)
	YC Core #5	**YC Core #6**
4th	Introduction to Distributed Systems (COM 3800)	Natural Language Processing (COM 3930)
	Machine Learning Applied (COM 4010)	Capstone Project (4020)
	Parallel Algorithms & Programming (COM 3820)	**YC Core #8 - ELECTIVE**
	YC Core #7 - ELECTIVE	

2. Y.C. Core and Honors Thesis Requirements

YC Core Requirements

- **All students need a total of 128 credits to graduate**
- **B.S. in C.S. students have the Y.C. Core requirements show in the table below.**
- **The B.S. in C.S. program has a residency requirement of 8 semesters, i.e. students must be full-time students in Y.C. for 8 semesters.**
- **If a student has 32 credits from the YU Israel program, the only courses he must take outside the CS major to reach 128 credits are the 8 YC Core classes listed below. In that case, we very strongly recommended that students take only one YC Core class each semester, in order to evenly distribute their workload across semesters.**

Requirement	Credits	Number of Courses
First Year Writing	3	1
BIB, JHI, JST, or JTP	4	2
Choose from Contemporary World Cultures (COWC) or Cultures Over Time (CUOT)	3	1
Interpreting the Creative (INTC)	3	1
Human Behavior and Social Institutions (HBSI)	3	1
Electives: choose from any of the following Y.C. departments: ART, BIO, BIB, CHE, ECO, ENG, HEB, HIS, JHI, JST, JTP, MUS, PHI, PHY, POL, PSY, SOC, SPA	6	2
TOTALS	**22**	**8**

For more details regarding what the various categories (COWC, HBSI, etc.) mean, what Y.C. courses count towards them, etc. please consult Y.C.'s web site or academic advising.

Combining CS Capstone and YC Honors Thesis

C.S. students have the option of combining their CS Capstone and Honors Thesis. (If you choose to do this, you may not take HON 4977H.)

The C.S. capstone requires each team to produce documents written for two audiences: skilled software engineers who want to understand or extend the system, and end-users who simply want to use the system. A student may produce a Y.C. honors thesis by writing a third, separate, document, which may be one of two types of paper:

1. Focusing on one aspect of the system produced in the capstone, write a literature review of the C.S. research & developments that led to the creation of the algorithms/approaches/technology used in that aspect of the system. This review should include a historical perspective on what existed before the approach was developed, how it was developed, what benefits or advantages it has over that which existed previously, and what the limitations of the approach are. The target audience for this

paper is people who are computer scientists themselves but are not familiar with this particular algorithm/approach/technology.

2. Fully explain one subsystem of the system produced in the capstone in a way that someone who is NOT a computer scientist can fully understand its inner workings. This will likely involve using diagrams, analogies, etc., to help the reader understand both the subsystem itself as well as the other elements in the system or environment (at the interface level only) that this subsystem must interact with. Although not perfectly analogous, students may draw some inspiration from Once Upon an Algorithm, How Computers Work, and Randall Munroe's various works.

Honors students combining the CS Capstone and honors thesis may register for 1 or 2 credits of Honors Thesis Writing in conjunction with the Capstone (COM 4020).

Each honors student must individually write his own honors thesis paper. Members of the same capstone team are required to choose different aspects / subsystems of their projects to focus on in this paper.

Scheduling and Supervision

1. The topic, and general approach, of the paper must be approved first by the student's capstone mentor and then by one other member of the C.S. faculty.
    a. The capstone mentor will double as the mentor for this paper.
    b. The mentor will coordinate with the department chair to have another CS faculty member review the proposal in a timely fashion.
2. The topic and approach must be submitted by the student no later than one week after his team's delivery of milestone 2 of the Capstone.

3. Course Descriptions

Preface to Course Descriptions 11
Introduction to Computer Science (4 credits) – COM 1300 11
Mathematics for Computer Science (4 credits) – COM 1310 12
Data Structures (4 credits) – COM 1320 12
Introduction to Algorithms (4 credits) – COM 2545 13
Design and Analysis of Algorithms (4 credits) – COM 2546 14
Computer Organization (4 credits) – COM 2113 15
Operating Systems – COM 3610 16
Networking - COM 2512 17
Cybersecurity – COM 4580 17
Programming Languages – COM 3640 18
Compilers and Tools – COM 3645 19
Introduction to Distributed Systems – COM 3800 20
Advanced Distributed Systems – COM 3810 21
Parallel Algorithms & Programming – COM 3820 22
Modern Data Management – COM 3580 22
Database Implementation – COM 3563 23
Machine Learning – COM 3920 24
Artificial Intelligence – COM 3760 24
Text Analysis and Natural Language Processing – COM 3930 25
Machine Learning Applied – COM 4010 26
Capstone Project – COM 4020 26

Preface to Course Descriptions

- Each course description includes
    - WHAT the course covers
    - OUTCOMES – what specific knowledge or skills you will gain in this course
    - WHY it is important for the major or track
- The course descriptions should make it clear to students what to expect from the course in general terms. Detailed syllabi for each course will come from the professor teaching the course, and the exact set of topics covered may vary from year to year.
- All courses are 3 credits (i.e. 3 hours of lecture a week) unless otherwise specified

Introduction to Computer Science (4 credits) – COM 1300

Prerequisites: None

WHAT:

This course is both an Introduction to "Computer Science" and an Introduction to "Programming". Students will get a sense of the sort of problems that can be solved using a computer, get experience writing computer programs that fulfill requirements which are described in English, and becoming skillful at iteratively improving a (possibly "buggy") program into a working program. Students will learn to apply computational thinking to frame problems and to guide the process of solving problems.

OUTCOMES:

- Students understand what Computer Science is (as opposed to "programming" alone)
- Students understand the tracks in the Computer Science major and what each prepares a student, and can make an informed decision regarding which track to take
- Students are able to apply computational thinking to map problems, and associated solution requirements, into programs that can be implemented and solved on a computers
- Students are able to write programs that meet a set of requirements.
- Students are able to test and debug small scale programs

WHY:

- Students must understand the basic contours of the industry to decide if they are interested in pursuing a career in it, and they must understand the tracks in the major and what each prepares a student for in order to make an informed decision about which track to pursue
- Code is the tool of a programmer. Before studying the science of C.S., students must learn how that tool works and how to use it properly. They also must internalize the basic patterns/styles of thought that programmers use to solve problems, regardless of what programming paradigm they happen to be working in.
- Basic competence in some programming language is needed in order to get anything done in computer science.

Mathematics for Computer Science (4 credits) – COM 1310

Prerequisites: Introduction to Computer Science (COM 1300)

WHAT:
The course will introduce students to a variety of topics in discrete mathematics that are essential for a Computer Science career. It emphasizes mathematical definitions and proofs as well as applicable methods. Topics include formal logic notation, proof methods; induction, well-ordering; sets, relations; elementary graph theory; asymptotic notation and growth of functions; permutations and combinations, and counting principles. The Python programming language will be introduced and assignments will require programming in Python.

OUTCOMES:
Students will:

- acquire a familiarity with the Python language and will use various Python software packages to solve problems in discrete mathematics.
- be able to understand mathematical reasoning in order to read, comprehend, and construct mathematical arguments
- understand the relationship between mathematical reasoning and the logical operations of Python.
- be able to construct logically correct proofs, especially those based on induction and well ordering.
- understand how inductive arguments are mirrored in recursive designs for computer programs.
- understand how sets, relations, and graphs are used for modeling in Computer Science and how Python data structures to represent them.
- be able to perform combinatorial analysis to estimate complexity
- understand the basics of probabilistic reasoning and its use in Computer Science

WHY:
The mathematical and analytical skills learned in this course are critical both for proper understanding of Algorithms and for success on tech interviews. Python has become the most popular language for data science and line-of-business coding, and thus studying these areas of math in the context of Python prepares students to compete for internships in these areas.

Data Structures (4 credits) – COM 1320

Prerequisites: Introduction to Computer Science (COM 1300). Corequisite: Mathematics for Computer Science (COM 1310)

WHAT:
Data Structures are logical constructs that facilitate organizing and accessing data efficiently. They are also the “raw materials” on which algorithms run. This course will cover at least the following:

- Introduction to growth rates, a.k.a. asymptotic analysis, as it pertains to the performance implications of choice of data structures
- Core data structure building blocks: arrays, lists, and recursion
- Basic data structures: hash tables, stacks, queues, dictionaries, trees (binary, 2-3, red-black, BTree, Tries, Heaps)

- Software engineering: Abstract Data Types, design by contract (preconditions, postconditions, invariants), assertions, pseudocode, and iterative development
- Using all the above in various applications

Students will complete a number of medium size (hundreds of lines of code each) programming assignments, which together will constitute the implementation of a single large system (thousands of lines of code). Each assignment involves the students 1) applying the software engineering skills learned in class and 2) using, and/or implementing from scratch, the data structures studied in class.

OUTCOMES:

- Students will be able to explain the characteristics, strengths, weaknesses, and when to use the most common data structures
- Students will be able to choose the right data structure(s) to use in order to meet or exceed a program's requirements
- Students will be able to implement programs that solve real-world problems making appropriate use of data structures
- Students will be able to independently learn about additional data structures
- Students will be able to articulate how data structures affect a program's performance
- Students will be able to articulate the relationship between data structures and algorithms
- Students will be able to apply software engineering skills in order to properly complete large projects in more advanced courses

WHY:

Data Structures are logical constructs that facilitate organizing and accessing data efficiently. They are also the "raw materials" on which algorithms run. Choosing the right data structures will make or break any non-trivial program because, together with the choice of algorithms, they dictate what a program can do in a reasonable amount of time. Software engineers must build complete, and efficient, programs that solve non-trivial problems. This requires making correct use of data structures as well as applying software engineering methodology to write software that meets requirements.

Introduction to Algorithms (4 credits) – COM 2545

Prerequisites: Data Structures (COM 1320), Mathematics for Computer Science (COM 1310)

WHAT:

Algorithms are methods for solving problems using a computer. This course is a survey of important algorithms that are both useful as well as prototypical in their design and performance characteristics. The course will analyze the algorithmic complexity of these important algorithms and derive basic lessons regarding algorithm choice and design. Specific topics include:

- Algorithm analysis, complexity & models of computation
- Efficient sorting and searching
- Graphs and graph algorithms
- Algorithm parallelization
- String sorting

OUTCOMES:

- Students will know the definition, and levels, of algorithmic complexity, and be able to analyze the complexity of an algorithm.
- Students will be able to apply basic algorithmic tools to design efficient algorithms.
- Students will be able to analyze a computational problem from an algorithmic perspective, and identify and employ appropriate algorithms and data-structures to solve these problems.

WHY:

While code is the tool of a programmer, algorithms are the methods with which a good programmer solves problems. A good programmer must be familiar with existing algorithms, understand their proper use, and be able to apply the lessons culled from their study when creating a new algorithm to solve a problem that existing algorithms do not address. Algorithmic thinking is so critical to good software engineering that it is often a large focus of tech interview processes.

Design and Analysis of Algorithms (4 credits) – COM 2546

Prerequisites: Introduction to Algorithms (COM 2545)

WHAT:

Algorithm design and analysis is fundamental to all areas of computer science. This course provides a rigorous framework for the study and analysis of algorithms. Building on both "Math for Computer Science" and "Introduction to Algorithms", this course focuses on techniques for the creation and understanding of efficient algorithms. Students will learn to analyze the computational complexity of a problem, recognize classes of problems, design new algorithms, and analyze proposed solutions. All concepts will be internalized via application to real-world problems. Specific topics include:

- Recurrences
- Greedy algorithms
- Divide-and-conquer algorithms
- Dynamic programming
- Network flow, max-flow, min-cut

OUTCOMES:

- Students will have mastered, and be able to apply, the mathematical foundation of analysis of algorithms
- Students will understand, and be able to apply, different algorithm design strategies and select the appropriate strategy for a range of problems.

WHY:

Real world problems can differ significantly from prototypical ones, and as such may require algorithmic solutions that are different than those available via well-known algorithms. As such, programmers must be prepared to create new solutions that are provably efficient. One must also be able to analyze available existing solutions to determine their acceptability in a given situation. While simplistic problems with small input sizes may not require such rigor, many modern applications have massive input sizes and strict performance requirements and therefore demand provably efficient solutions.

Computer Organization (4 credits) – COM 2113

Prerequisites: Introduction to Computer Science (COM 1300)

WHAT:

Computer science has constructed layers of abstractions which allow us to write code at a much higher level than assembly language (or even than C.) In this course students learn the basic concepts underlying all computer hardware systems, get a look “under the hood” to see how these abstractions are translated down to actual instructions that can be executed on computers, and how computer systems actually execute programs and store information. Specific topics include:

- Introduction to the C programming language and Linux command line
- Finite binary representations of integers and real numbers, both scalars and vectors.
- X86_64 assembly language
- In-memory layout of executing C and assembler programs.
- Cybersecurity implications of the X86_64 architecture and program execution environment.
- Memory hierarchies and the importance of temporal and spatial locality.
- Signaling asynchronous events; blocking and delaying signals
- Issues in implementing I/O correctly
- Principles of low-level code optimization
- Dynamic memory allocation schemes.

OUTCOMES:

- Students understand, and are able to debug, (unoptimized) assembly language code produced by current C compilers
- Students will have a basic intuition for what assembly language code will be generated by compilers from their high level code.
- Students will be able to predict and simulate the effects of cache memories on running programs for both hypothetical and actual computer hardware
- Students will understand how Linux programs process asynchronous signals, and how to write signal-safe code.
- Students will understand how virtual memory affects program performance on X86_64 processors
- Students will understand how computer architecture gives rise to security flaws
- Students will be able to apply all the knowledge described above to write better high level code.

WHY:

The abstractions on top of which we commonly write code are not perfect, and at times their imperfections result in critical performance or resource management problems. Knowing how computers actually work "under the hood" gives a programmer the tools needed to avoid and/or address such problems. This course also serves as a foundation for the study of operating systems and parallel programming, where an understanding of systems-level issues is required.

Operating Systems – COM 3610

Prerequisites: Computer Organization (COM 2113), Data Structures (COM 1320)

WHAT:

The operating system of any computing device (PC, smartphone, server, etc.) manages the resources of the computer and provides applications with a software interface to the computer's hardware resources. This course builds on the knowledge students gained regarding the hardware-software interface in Computer Organization and examines the roles and architectures of modern operating systems. The operating system is responsible for allowing resources (e.g., disks, networks, and processors) to be shared, providing common services needed by many different programs (e.g., file service, the ability to start or stop processes, and access to the printer), and protecting individual programs from one another. The knowledge gained in this class is important in areas such as parallel programing, compilers, distributed systems, databases, and networking. Specific topics include:

- Operating System structure
- Processes, threads, kernel space, and user space
- Deadlock, livelock, starvation, and fairness
- Memory management
- File systems
- Virtualization
- Multiprocessors
- Operating systems security principles

OUTCOMES:

- Students understand the history, necessity, and major functions of operating systems
- Students are able to write system level software that uses or replicates (as appropriate) common features of modern operating systems
- Students are able to write and debug properly synchronized multi-threaded code at both the process and thread levels in Linux, preventing both deadlock and starvation
- Students can create, configure, and manage virtual machines using a hypervisor
- Students understand, and can configure and use, the security-related features of modern operating systems

WHY:

Knowing how to properly manage the various resources of a computer (memory, files, CPU, etc.) is a key area of expertise for any programmer, and operating systems is the context in which one learns about resource management. One who does not know how an operating system manages resources neither understands what happens when code is executed nor how to write code that will perform its job efficiently on any modern computer.

Networking - COM 2512

Prerequisites: Operating Systems (COM 3610)

WHAT:
Internet access is taken as a given in the developed world. This course explains the principles and practices of computer networking in general, and specifically introduces how the internet, and the applications and services built on top of it, work. The goal is for students to learn not only how the internet, and the services on top of it, work today, but also why they were designed this way, and how to build high quality networked software systems. Topics include:

- History of networking; protocol stacks before the Internet
- The most common Application Level protocols and APIs
- Connection-oriented and connectionless transport
- The two "planes" of the routed Internet (control and data)
- The wide varieties of link-level technology (Ethernet, WiFi, etc.)
- Wireless and Mobile networks, including high mobility and secure access.
- Basic application of cryptography for secure network communication
- Cybersecurity attacks and defenses for computer networks

OUTCOMES:

- Students understand the behavior and architecture of the networks protocols that make up the internet
- Students will be able to articulate the principles of network security
- Students will be able to program networked services and applications
- Students will be able to build reliable services from unreliable ones

WHY:
From simple mobile and web applications to massive scale distributed systems, most applications and systems today make use of the internet. One must understand the capabilities and limits of this architecture in order to build high quality systems on top of it.

Cybersecurity – COM 4580

Prerequisite: Operating Systems (COM 3610), Introduction to Algorithms (COM 2545), Networking (COM 2512)

WHAT:
This course covers the aspects of cybersecurity that are critical for building and deploying secure applications in today's highly networked and distributed technology environment. Students will study the major aspects of system security (authentication, access control, attacks and defenses, etc.), as well as cryptography and secure coding. Specific topics include:

- Cryptography
- Operational Security
- Authentication, Authorization, and Access controls

- Malware
- Web-based vulnerabilities
- Designing and coding secure software
- Network security

OUTCOMES:

- Students will be able to write secure software, as well as revise poorly written software to mitigate common security vulnerabilities
- Student will be able to identify and use the appropriate cryptographic tools when writing and deploying software
- Students understand, and will be able to function effectively, both on the offensive ("Black Hat") and defensive ("White Hat") sides of Cybersecurity
- Students will be able to design and configure simple network defense strategies with firewalls and routers
- Students will be able to build and analyze threat models for specific security weaknesses

WHY:
Most applications and systems today of all sizes (from mobile apps to corporate data centers) are connected to the internet, which means that they are subject to cyberattacks. As such, billions of dollars will be made and/or lost in the arena of cybersecurity. The pervasiveness of threats and increasingly extreme costs (both immediate and reputational) of a system being compromised means that security can no longer be an afterthought in software engineering. Understanding cybersecurity and writing secure code is a critical skill which students learn in this course.

Programming Languages – COM 3640

Prerequisites: Introduction to Algorithms (COM 2545)
WHAT:
Programming Languages define the syntax and semantics that are available to encode logic that will be executed by a computer. Many choices and trade-offs are made when designing a language, when deciding what language to use when writing a program, and when deciding which parts of a language to use for a given program. This course introduces formal approaches for defining a language, surveys popular programming languages and paradigms and examines their designs, trade-offs, and proper use. Specific topics include:

- Theory of Computation foundations for programming languages
- Core Issues in Language Design
- Describing Language Syntax and Semantics
- Programming Paradigms: Imperative, Object Oriented, and Functional.
- Programs at Runtime
- Programming in the various paradigms using various languages (e.g. JavaScript, Java, Python), highlighting the orthogonality of paradigms and languages
- In-depth examination of at least one language, including language history, features, and runtime

OUTCOMES:

- Students can articulate the difference between syntax and semantics, and can specify each
- Students can articulate the tradeoffs between language paradigms as well between languages
- Students can make well informed choices when deciding what language, and language features, to use when writing a given program to solve a given problem
- Students can write programs in a range of languages and using a range of language features

WHY:

In any profession, the better one understands the tools of the trade, the better a craftsman one will be. In computer science, programming languages are the tools of the trade and thus demand study. Furthermore, most software development organizations make use of multiple languages, using each language to complete the tasks for which it is best suited. As such, one must be familiar with a range of programming paradigms and languages. Lastly, new languages appear frequently but very few are widely used. One must be able to differentiate between hype and substantive innovation to avoid thrashing on learning short-lived languages.

Compilers and Tools – COM 3645

Prerequisites: Programing Languages (COM 3640)

WHAT:

Compilers and interpreters are the “magical” programs that take the text we refer to as “code” and transform it into something that controls the behavior of a computer. While tremendous amounts of effort and rigor go into designing high-level languages that deliver tremendous productivity gains over their low level equivalents, at runtime all code is transformed into a lower level representation in order to execute. To write high quality software, a software engineer must understand the steps and results of this transformation. Achieving this understanding is the focus on the majority of this course.

Large programs are often written by many programmers across multiple organizations. Therefore, the build process for real-world programs requires more than just a compiler or interpreter; build tools (e.g. Ant, Maven, Gradle) are used to manage the process of transforming code bases into executables. In addition, some applications' delivery cycles and/or required domain expertise result in situations where creation of a DSL (domain specific language) is the overall fastest approach to application development. DSLs, however, require tool chains just as standard programming languages do, hence the need for software tools to create those tools. The last section of this course focuses on these various types of tools.

Specific topics will include:

- Parsing and lexical analysis
- symbol tables
- semantic analysis
- intermediate representations
- code generation
- run-time organization and memory management

- parser generators
- compiler infrastructures
- build managers
- configured & generated language tools

OUTCOMES:

- Students understand how code gets transformed into machine instructions, and can apply that knowledge to make good software design choices
- Students know, and are able to apply, best practices and standard tools for creating domain specific languages

WHY:
Understanding how code gets transformed into operations on a computer will help a programmer write better code. Understanding how compilation/building of code gets scaled up to multi-module cross-team software is essential in modern application development. The ability to invent or adopt a DSL and create the required tools to facilitate efficient use thereof will enable one to solve real-world problems that are not cleanly addressed by existing languages or tools.

Introduction to Distributed Systems – COM 3800

Prerequisites: Introduction to Algorithms (COM 2545), Operating Systems (COM 3610). Corequisites: Parallel Algorithms & Programming (COM 3820).

WHAT:
Distributed systems enable the aggregation of many networked computers to construct highly available and scalable services. This course will introduce the core challenges of, and approaches to, building distributed systems. Aspects of cloud computing will be examined in some depth. Specific topics include:

- time synchronization
- coordination, consensus
- scalability, replication
- availability, fault tolerance
- cluster scheduling
- cloud computing architecture and usage

OUTCOMES:

- Students will be able to clearly articulate the value of distributed systems.
- Students will be able to clearly articulate the key challenges to overcome when building a distributed system.
- Students will be able to apply well established approaches to solving the key challenges when building distributed systems.
- Students will have experience solving at least one key distributed systems challenge from scratch, and thus have the necessary experience to address such challenges in the future.
- Students will be able to use, and explain, common cloud computing building blocks.

WHY:

There has been an explosion of applications of distributed systems, with horizontal scalability enabling everything from Google's search engine to Amazon Web Services to breakthroughs in deep learning and other aspects of data science. Cloud computing has enabled organizations of all types and sizes to build large scale distributed systems, and thus is being leveraged across many industries. Practitioners must have an accurate understanding of the uses, strengths, and challenges of distributed systems in order to function in today's top industries.

Advanced Distributed Systems – COM 3810

Prerequisites: Introduction to Distributed Systems (COM 3800)

WHAT:

Building on the introductory course, this course provides a deeper understanding of distributed systems and focuses on the implementation and maintenance of scalable, fault-tolerant distributed systems. Specific topics include:

- Distributed systems at runtime: instrumentation, monitoring
- System performance: defining criteria, predicting, guaranteeing (SLAs.) Performance of distributed systems.
- Detecting and remedying failure scenarios.

OUTCOMES:

- Students will be able to instrument and monitor distributed systems.
- Students will be able to build a fault tolerant distributed system.
- Students will be able to build a distributed system whose performance is well understood and predictable.
- Students will be able to use, and will gain understanding of and experience with, a set of distributed system architectures that are widely used today.

WHY:

The massive growth in both the volume of data and the varied ways it is used (mobile gaming, financial data/applications, deep learning, predictive modeling, real time ad exchanges, etc.) has made it impossible to implement many high-value applications on a single computer. Many companies across industries have come to understand that significant benefits can be realized by sharing both computing resources and data sets across teams/departments/etc. All of these applications, companies, and industries need computer scientists who are sufficiently knowledgeable in distributed systems to build scalable, fault-tolerant distributed systems. Without scalability and fault-tolerance, distributed systems can’t be used for mission critical applications.

Parallel Algorithms & Programming – COM 3820

Prerequisites: Design & Analysis of Algorithms (COM 2546), Operating Systems (COM 3610)

WHAT:
This course will examine basic choices and tradeoffs made in parallel systems, with a strong focus on concurrent programming and parallel algorithm design. Specific topics include:

- concurrent programming in a high-level language (e.g. Java)
- parallel algorithm design techniques
- locality
- implicit vs. explicit parallelism
- shared vs. non-shared memory
- synchronization mechanisms (locking, atomicity, transactions, barriers) within a single system and across systems
- parallel programming models (threads, data parallel/streaming, futures, message passing, transactions)

OUTCOMES:

- Students will be proficient in the areas of parallelism, concurrency, locality and distribution in the context of algorithm design and implementation
- Students will be able to identify which parallel strategy is appropriate to solve a given problem

WHY:
Whether running multiple threads on one computer or distributed across many computers, parallelism is widely used in modern server applications. Using threads or other concurrent mechanisms without understanding them almost inevitably leads to unexpected and/or wrong results, in addition to hard to debug problems. This course gives students fluency in the core issues and techniques used in scalable server side systems today.

Modern Data Management – COM 3580

Prerequisites: Design & Analysis of Algorithms (COM 2546)
WHAT:
This course surveys the wide range of database types widely used today, what they each are good at, and how to combine various types into a data pipeline. It starts with a rigorous but fast-paced introduction to the relational model, schemas, indices, views, SQL, and transactions. It then proceeds to cover a number of other types of data systems, giving the student a broad based view of modern data management. Specific topics include:

- The relational model, SQL, database normalization, and relational database management systems
- NoSQL databases, including key-value stores, document stores, column stores and graph databases
- Streaming and messaging systems
- Unified, large-scale, data processing tools such as Apache Spark
- Understand tradeoffs between implementing business logic on the client versus and the server, and how to integrate business logic with a back-end datastore

OUTCOMES:

- Students will understand the conceptual model of relational databases, and how to use the features of a relational database management system (RDBMS).
- Students will be knowledgeable about differences between SQL and various types of NoSQL databases, and will be able to articulate what use cases motivate the selection of one type of database versus another
- Students will be able to build systems using a number of types of database systems
- Students will be able to create and manage data pipelines across multiple types of data systems

WHY:
In the real world, data doesn't fit into one schema or system fully (e.g., relational only or NoSQL only). Typically, enterprise data reside in multiple data store instances and in multiple types of data stores. In order to understand, as well as properly use and integrate enterprise data, software engineers must be able to construct data pipelines that can apply data transformations and integrations to these different types of data-stores. In addition, software engineers must understand which type of data-store to use when building a new data source.

Database Implementation – COM 3563

Prerequisites: Design & Analysis of Algorithms (COM 2546), Operating Systems (COM 3610), Modern Data Management (COM 3580)

WHAT:
This course focuses on the architecture and implementation of relational databases. Specific topics include:

- Database system architecture
- Storage organization, access, and buffer management
- Storage data structures
- Query planning and execution
- Transaction management
- Recovery
- Concurrency control.

OUTCOMES:

- Students will understand the architecture and implementation of relational databases.
- Students will understand the importance of, and be familiar with at least one implementation approach to, DBMS system features such as storage organization, buffer management, indices, logging, and query planning & optimization.
- Students will understand and be able to apply the principles of transaction processing (atomicity and isolation) to build a small-scale transactional database system.

WHY:

Databases are essential to every business. All non-trivial applications depend on data, and thus in turn on well-designed databases. A backend systems engineer needs a deep understanding of the challenges and approaches in implementing a database system in order to be able to build high quality server side systems that use databases to their fullest potential. While not all data will be stored in relational databases, all systems that store data must be designed by computer scientists who understand the challenges and issues that relational databases attempt to solve.

Machine Learning – COM 3920

Prerequisites: Design & Analysis of Algorithms (COM 2546), Mathematical Statistics (MAT 2462), Artificial Intelligence (COM 3760)

WHAT:

Machine learning's goal is to develop applications whose accuracy in predicting the value of unknown data improves by examining more and more known data. This course introduces the main principles, algorithms, and applications of machine learning, as well as important open source libraries and cloud-based machine learning services that practitioners are using to build real systems. Specific topics will include:

- Overview of Machine Learning (ML)
- Logistic Regression, Softmax
- Neural Networks, Convolutional Neural Networks
- Decision Trees and Ensembles
- NLP
- Reinforcement Learning
- Unsupervised Learning

OUTCOMES:

- Students are able to implement and analyze ML algorithms
- Students are able to describe the formal properties of models and algorithms for ML and explain the practical implications of those results
- Students are able to select and apply the most appropriate ML method to solve a given learning problem

WHY:

Predictions generated by machine learning systems are used today in many applications across many industries. The quality and adoption of machine learning has increased dramatically. Students who wish to compete for jobs in top tech firms and other data-driven fields must be competent in machine learning.

Artificial Intelligence – COM 3760

Prerequisites: Design & Analysis of Algorithms (COM 2546)

WHAT:

Artificial Intelligence is the science of representing knowledge and making good decisions based on a set of incomplete information, i.e. under uncertainty. The course begins by placing Artificial Intelligence (AI) in the broader context of popular culture, Philosophy of Mind, and Cognitive Psychology. It then goes to in-depth treatments of methods for automated reasoning, automatic problem solvers and planners,

knowledge representation mechanisms, game playing, machine learning, and statistical pattern recognition. Specific topics include:

- The broad context of Artificial Intelligence
- General capabilities of Intelligent Agents
- Problem Solving, including: solving by search, constraint satisfaction, game playing
- Knowledge & Planning: knowledge representation, inferencing, generating sequences of actions
- Uncertainty: quantifying, reasoning, and decision making
- Machine Learning in the context of Artificial Intelligence

OUTCOMES:

- Students will be able to explain key terms about artificial intelligence including differences
- between machine learning (ML), GOFAI ("good old fashioned AI"), and terms in key ML and AI
- Students will attain mastery, and be able to fully explain, key concepts in artificial intelligence, including heuristic search, game playing, formal logic, knowledge representation, planning, decision theory, machine learning, and natural language processing.
- Students will be able to use/apply key Artificial Intelligence concepts in code

WHY:

Companies in information-rich industries, such as tech, finance, and marketing, as well as industries that can get reams of information from sensors, such as the automotive industry, are increasingly focused on systems that can make decisions when faced with uncertain and/or incomplete information, which is where AI shines.

Text Analysis and Natural Language Processing – COM 3930

Prerequisites: Machine Learning (COM 3920)

WHAT:

This course examines computational methods for analyzing human language textual data in order to detect meaning and extract information. Applications of these methods include sentiment analysis, information retrieval, and trend prediction. Specific topics include:

- What is natural language processing and the challenge of doing it computationally
- Major tasks that NLP undertakes
- Uses and limitations of n-gram analysis
- Using NLP and syntactical analysis for text mining
- Understanding and implementing search engines
- Using and implementing modern word embedding techniques such as Word2Vec

OUTCOMES:

- Students will be able to articulate the fundamentals of natural language processing
- Students will be able to competently use several major software packages for NLP
- Students will be able to apply machine learning for text analysis
- Students will be able to implement Information Retrieval Algorithms
- Students will be able to implement and use Word Embedding algorithms such as Word2Vec

WHY:
Vast amounts of information is created in the form of unstructured data – web pages, social media posts, emails, presentations, analysts' reports, news content, etc., and companies in many industries are devoting resources to extracting valuable information therefrom. The ability to extract useful information from such data sources is therefore a critical tool of an AI-focused software engineer.

Machine Learning Applied – COM 4010

Prerequisites: Machine Learning (COM 3920). Corequisite: Introduction to Distributed Systems (COM 3800)

WHAT:
"Applied machine learning", sometimes called Machine Learning Engineering, is the implementation of machine learning algorithms and the productionization of a system or product that uses the models. This course focuses on learning applied skills that enable you to build and deploy into production real-world ML applications. Specific topics include:

- Machine Learning lifecyle
- Model deployment
- Big data's use in ML
- Real-world ML applications

OUTCOMES:

- Students are able to apply all that they learned in the prerequisite and corequisite classes to build a complete ML solution
- Students are able to apply software engineering concepts and best practices when building the aforementioned solutions
- Students deeply understand complete solutions to a number of applications of ML
- Students have specified, and received approval for, their Capstone project (COM 4020)

WHY:
This course helps the student initially integrate all that he has learned in the AI track, sets the student up for the Capstone project, and provides an initial demonstration of the level of the student's AI competence to prospective employers and/or graduate schools.

Capstone Project – COM 4020

Prerequisites: Machine Learning Applied (COM 4010) ***OR*** *Advanced Distributed Systems (COM 3810)*

WHAT:
Building a complete system that meets all functional requirements and is technically sound is a challenging, multifaceted, and iterative process which requires one to bring to bear all that he has learned throughout his CS education. Under the supervision of faculty, students will complete a realistic software engineering project that solves a real-world problem and meets or exceeds all relevant functional and technical criteria. With faculty approval, students (including students from different tracks in the major) may collaborate on building a single system in which they each develop different aspects of the solution/system.

OUTCOMES:

- Students understand the challenges inherent in, and correct approaches to, building a complete, end-to-end system.
- Students are able to successfully follow a complete Software Engineering Lifecycle

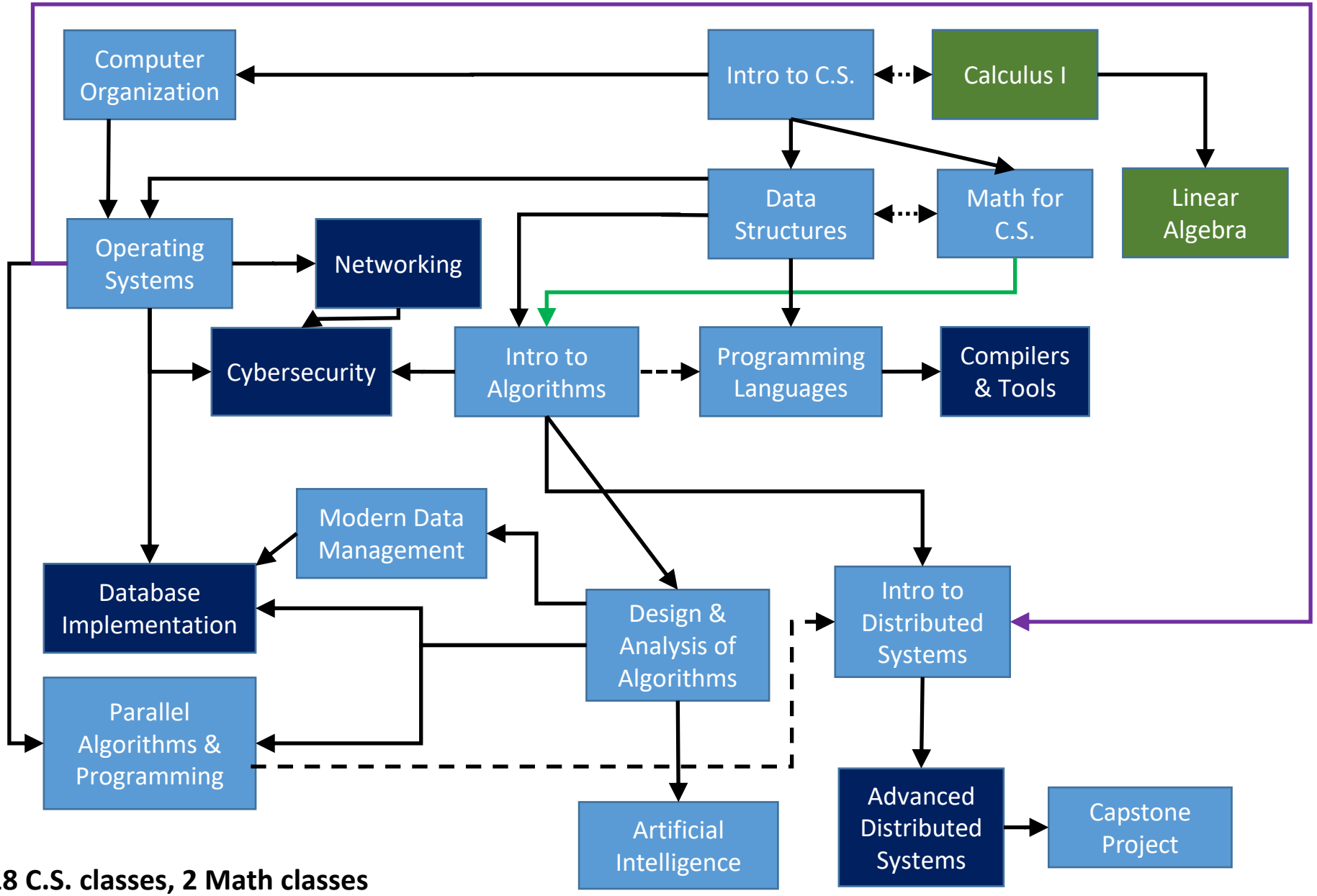
WHY:
The senior project will facilitate the student further internalizing and integrating all the he has learned in in the C.S. major and prepare him for “professional-grade” success. It also demonstrates to prospective employers or graduate schools the level of the student’s competence.

4. Prerequisite Flow Charts

The following pages have prerequisite flow charts for each of the tracks in the major.

Prerequisite Flow Charts

B.S. in C.S. - Distributed Systems Track

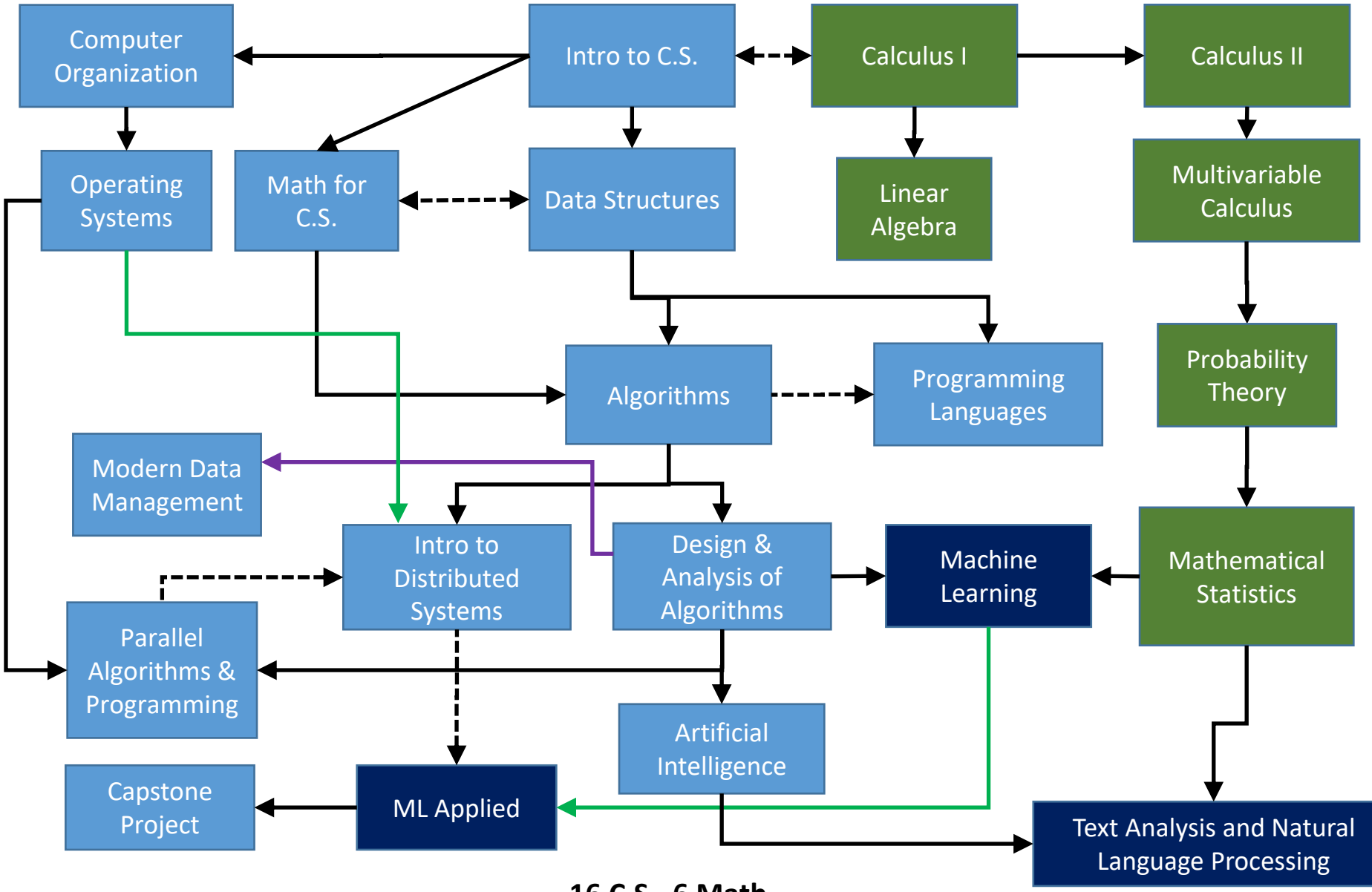


18 C.S. classes, 2 Math classes

Dark Blue indicates not required for Artificial Intelligence track.

Dotted line: Corequisite. Solid Line: Prerequisite. Different color arrows are used when lines cross to make it clear.

B.S. in C.S. – Artificial Intelligence Track



16 C.S., 6 Math.

Dark Blue indicates not required for Distributed Systems track.

Different color arrows are used when lines cross to make it clear.

Faculty

(In alphabetical order)

Faculty Statistics

- **153+ years of full-time corporate experience** across Amazon, Citi, Goldman Sachs, Google, IBM, Intel, and others
- **69 issued U.S. patents**
- **200+ publications**

Judah Diament

Professor, Department Chair

- **IBM T.J. Watson Research Center: 2000-2014**
  - **Patents:** 14 U.S. patents issued
  - **Publications:** 5 conference papers, 1 journal article
  - Impacted multiple IBM software products, including shipping code
- **Goldman Sachs: 2014-2016**
  - Vice President, Finance Engineering
- **Alumnus of Y.U., N.Y.U. (M.S. in C.S.), R.I.E.T.S.**
- Judah's LinkedIn page
- diament@yu.edu

Dave Feltenberger

Adjunct Professor

- **Prof. Feltenberger currently teaches: Machine Learning, Machine Learning Applied, and AI Capstone Project**
- Prof. Feltenberger's professional background:
- **Google, 2012-Present: Principal Engineer**, Technical lead for Semantic Location. Previously - Senior Staff Software Engineer, Quality & ML in Google Maps; founder of Corp Eng ML team
- **Goldman Sachs: 2010-2012** Senior Software Engineer, post-execution trading platform

Peter Grabowski
Adjunct Professor

- **Prof. Grabowski currently teaches Machine Learning Applied in Yeshiva College and is on the Data Science faculty of University of California, Berkeley.**
- Prof. Grabowski's professional background:
- **Google, 2017-Present:** Professor Grabowski currently leads the Core ML's Gemini applied research team. He previously led Google's Enterprise ML team.
- **Nest, 2014-2017:** founded data integration and ML team

Avraham Leff

Professor

- **PhD, Computer Science, Columbia University: 1992**
- **IBM T.J. Watson Research Center: 1991-2017**
  - **Patents**: 21 U.S. patents issued
  - **Publications**: 45 conference papers & journal article
  - Impacted multiple IBM software products, including shipping code
- Avraham's LinkedIn Page

Ramesh Natarajan

Adjunct Research Professor

- **PhD, University of Texas at Austin**
- **Google, 2020-2023:** Google Cloud, Software Engineer and Tech Lead
- **Amazon, 2014-2020 :** Research Scientist
- **IBM, 1988-2014:** Research Staff Member
- **Patents:** 25 granted (at IBM, Amazon and Google). IBM High-Value Patent Award.
- **Ramesh's LinkedIn Page**

Avi Rosenfeld

Adjunct Professor

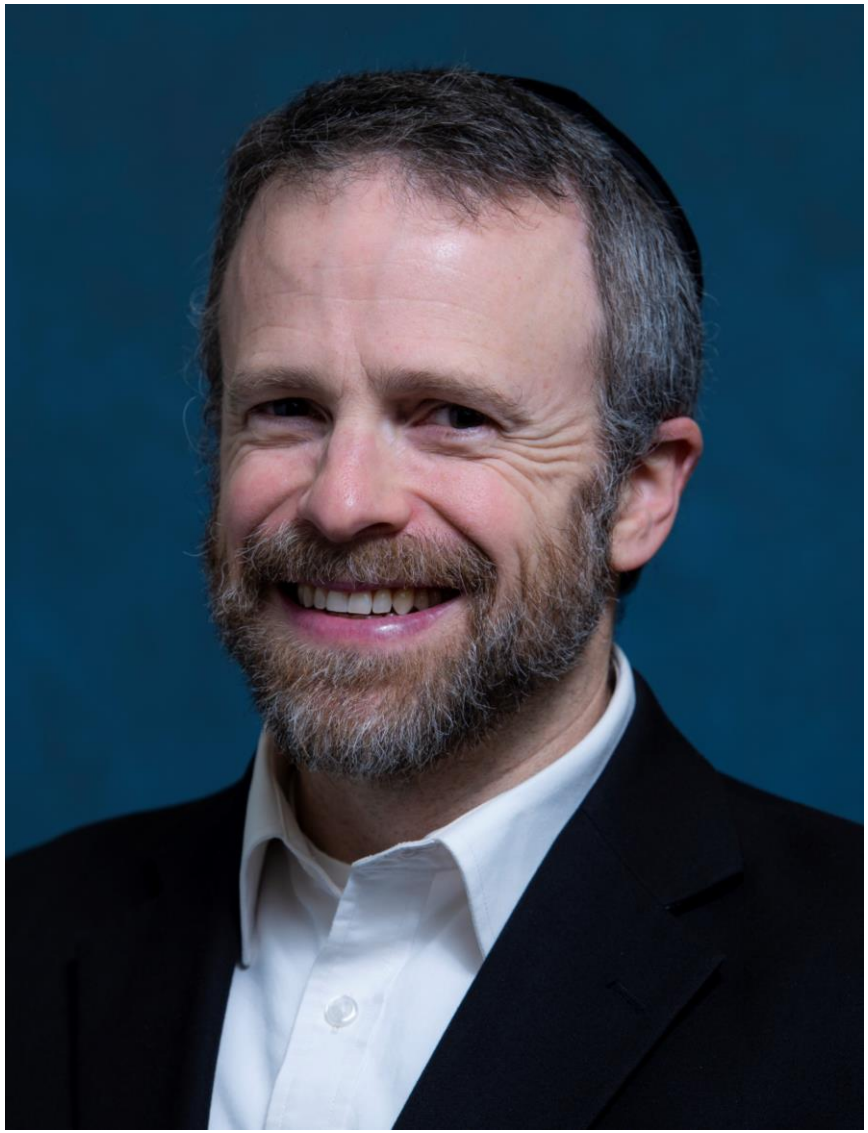
- **PhD, Computer Science / Artificial Intelligence, Bar Ilan: 2007**
- **Associate Professor, Machon Lev, Jerusalem**
  - Head of Data Science Program
  - Publications: 80+
  - Patents: 3
- **One of four member of Israel's Education Counsel responsible for judging all academic degrees in Data Science**
- **Alumnus of MTA, YC, RIETS, Azrieli**
- **Avi's LinkedIn Page**

Akiva Sacknovitz

Clinical Professor

- **Citigroup: 2010-2022**
  SVP, Global Spread Products, Securitized Markets IT
  Led the design and implementation of a fault-tolerant messaging and service API framework and a distributed queueing system to support front-office desk pricing and end-of-day risk calculations.
- **Credit Suisse: 2004-2010**
  Credit Derivatives, pricing and risk applications
- **Shopping.com (eBay): 2003-2004**
  Research engineer, deal discovery and classification
- **Network Analysis Center: 1996-2003**
  Wide-area network analysis software development
- **Alumnus of Y.U., N.Y.U. (M.S. in C.S.), R.I.E.T.S.**
- **Akiva's** LinkedIn page

Ben Wymore
Clinical Professor



- **M.S. in C.S., University of Minnesota: 1997**
- **Intel Research: Software Engineer**
- **Crestron Electronics: Senior Software Engineer & Team lead**
- **Patents**: 9 U.S. patents issued