

[Unit 1 - Programming Skills and Computer Organization](#)

[Unit 2 - Data Structures and Efficiency](#)

[Unit 3 - Scaling Up Computing](#)

[Unit 4 - CS as a Tool](#)

[Unit 5 - CS in the World](#)

Unit 1 - Programming Skills and Computer Organization

- Understand the **expectations**, **resources**, and **policies** associated with 15-110
- Define the essential components of computer science, **algorithms** and **abstraction**
- Construct **plain-language algorithms** to solve basic tasks
- Recognize and use the basic **data types** in programs
- Interpret and react to basic **error messages** caused by programs
- Use **variables** in code and trace the different values they hold
- Understand how different **number systems** can represent the same information
- Translate **binary numbers** to decimal, and vice versa
- Interpret binary numbers as abstracted types, including **colors** and **text**
- Identify the **argument(s)**, **returned value**, and **side effect(s)** of a function call
- Use **function definitions** when reading and writing algorithms to implement procedures that can be repeated on different inputs
- Recognize the difference between local and global **scope**
- Trace the **call stack** to understand how Python keeps track of nested function calls
- Use **libraries** to import functions in categories like math, randomness, and graphics
- Recognize that the process of **tokenizing**, **parsing**, and **translating** converts Python code into instructions a computer can execute
- Recognize how the different types of **errors** are raised at different points in the Python translation process
- Use **logical operators** on Booleans to compute whether an expression is True or False
- Use **conditionals** when reading and writing algorithms that make choices based on data
- Translate Boolean expressions to **truth tables** and **circuits**
- Translate **circuits** to **truth tables** and Boolean expressions
- Recognize how addition is done at the circuit level using **algorithms and abstraction**

- Use **while loops** when reading and writing algorithms to repeat actions while a certain condition is met
- Identify **start values**, **continuing conditions**, and **update actions** for **loop control variables**
- Use **for loops** when reading and writing algorithms to repeat actions a specified number of times
- Recognize which numbers will be produced by a **range** expression
- Translate algorithms from **control flow charts** to Python code
- Use **nesting** of statements to create complex control flow

Unit 2 - Data Structures and Efficiency

- **Index** and **slice** into strings to break them up into parts
- Use for loops to loop over strings by **index**
- Use **string operations and methods** to solve problems
- Read and write code using **1D** and **2D lists**
- Use **list methods** to change lists without variable assignment
- Recognize whether two values have the same **reference** in **memory**
- Recognize the difference between **mutable** vs. **immutable** data types
- Recognize the difference between **destructive** vs. **non-destructive** functions/operations
- Use **aliasing** to write functions that destructively change lists
- Define and recognize **base cases** and **recursive cases** in recursive code
- Read and write basic **recursive code**
- Trace over recursive functions that use **multiple recursive calls** with Towers of Hanoi
- Recognize **linear search** on lists and in recursive contexts
- Use **binary search** when reading and writing code to search for items in sorted lists
- Identify the **worst case** and **best case** inputs of functions
- Compare the **function families** that characterize different functions
- Calculate a specific function or algorithm's efficiency using **Big-O notation**
- Identify the **keys** and **values** in a dictionary
- Use **dictionaries** when writing and reading code that uses pairs of data
- Use **for loops** to iterate over the parts of an **iterable** value

- Recognize the requirements for building a good **hash function** and a good **hashtable** that lead to **constant-time search**
- Recognize and trace the algorithms for **selection sort** and **merge sort**
- Compute and contrast the efficiency and Big-O runtimes of **selection sort** and **merge sort**
- Identify core parts of **trees**, including **nodes**, **children**, the **root**, and **leaves**
- Use **binary trees** implemented with dictionaries when reading and writing code
- Identify core parts of **graphs**, including **nodes**, **edges**, **neighbors**, **weights**, and **directions**.
- Use **graphs** implemented as dictionaries when reading and writing simple algorithms in code
- Identify whether a tree is a **binary search tree**
- Search for values in BSTs using **binary search**
- Analyze the **efficiency** of binary search on a **balanced** vs. **unbalanced** BST
- Search for paths in **graphs** using **breadth-first search** and **depth-first search**
- Analyze the **efficiency** of BFS and DFS on a graph
- Identify **brute force approaches** to common problems that run in $O(n!)$, including solutions to **Travelling Salesperson** and **puzzle-solving**
- Identify **brute force approaches** to common problems that run in $O(2^n)$, including solutions to **subset sum** and **exam scheduling**
- Define whether a function family is **tractable** or **intractable**
- Define the complexity classes **P** and **NP**, and explain why they are important

Unit 3 - Scaling Up Computing

- Define and understand the differences between the following types of concurrency: **circuit-level concurrency**, **multitasking**, **multiprocessing**, and **distributed computing**
- Create **concurrency trees** to increase the efficiency of complex operations by executing sub-operations at the same time
- Recognize certain problems that arise while multiprocessing, such as **difficulty of design** and **deadlock**
- Create **pipelines** to increase the efficiency of repeated operations by executing sub-steps at the same time
- Use the **MapReduce pattern** to design and code parallelized algorithms for distributed computing

- Recognize core terms related to the internet, including: **browsers, routers, ISPs, IP addresses, DNS servers, protocols, packets, and cloud**
- Understand at a high level the **internet communication process** that happens when you click on a link to a website in your browser.
- Understand at a high level that the internet is **fault tolerant** due to being **distributed**
- Define the following terms: **data privacy, data security, authentication, and encryption**
- Recognize the traits of the internet that make it more prone to **security attacks** and recognize common security attacks (**DDOS** and **man-in-the-middle**).
- Trace common **encryption** algorithms, such as the **Caesar Cipher** and **RSA**, and recognize whether they are **symmetric** or **asymmetric**
- Evaluate the efficiency of **breaking** encryption algorithms based on **keyspace**.

Unit 4 - CS as a Tool

- Use the **input** command and **try/except** structures to handle direct user input in code
- Implement and use **helper functions** in code to break up large problems into solvable subtasks
- Install **external modules** with the **pip** command
- Read and write data from **files**
- Interpret data according to different protocols: **plaintext, CSV, and JSON**
- **Reformat** data to find, add, remove, or reinterpret pre-existing data
- Represent the state of a system in a **model** by identifying **components** and **rules**
- **Visualize** a model using graphics
- Update a model over **time** based on **rules**
- Update a model after **events** (mouse-based and keyboard-based) based on **rules**
- Given a dataset, identify **categorical, ordinal, and numerical** features which may help predict the correct output for a given input
- Identify how the three major categories of learning (**supervised, unsupervised, and reinforcement**) interact with three major categories of reasoning (**classification, regression, and clustering**) and decide which type of learning / reasoning best fits a problem statement.
- Describe how **training, validation, and testing** are used to build a model and measure its performance

- Perform **basic analyses** on data, including calculating **statistics** and **probabilities**, to answer simple questions
- Choose an appropriate **visualization** to create based on the number of **dimensions** and **data types**
- Create simple **matplotlib visualizations** that show the state of a dataset using **APIs** and **examples**
- Use **Monte Carlo methods** to estimate the answer to a question
- Organize **animated simulations** to observe how systems evolve over time
- Recognize how AIs attempt to achieve **goals** by using a **perception, reason, and action** cycle
- Build **game decision trees** to represent the possible moves of a game
- Use the **minimax algorithm** to determine an AI's best next move in a game
- Design potential **heuristics** that can support 'good-enough' search for an AI

Unit 5 - CS in the World

- Big Ideas of: Introduction of the **theoretical concept** of a computer
- Big Ideas of: Construction of the first computer **hardware and software**
- Big Ideas of: Transition of computers from government/corporate to **personal**
- Big Ideas of: Connection of computers via the **internet**
- Understand the current extent of **data collection** on the internet and how data is used
- Recognize the uses and drawbacks of **facial recognition** algorithms in different contexts
- Identify the societal impact when **automated decision making** replaces human decision making due to the explainability problem and job displacement
- Recognize and describe the key impacts of future computing ideas, including: **cryptocurrencies, NFTs, 5G, deepfakes, virtual reality, and quantum computing.**