Unit 1 - Programming Skills and Computer Organization

- 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**

- Use **function calls** to run pre-built algorithms on specific inputs
- Identify the **argument(s)** and **returned value** of a function call
- Use **libraries** to import functions in categories like math, randomness, and graphics

- 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 **function calls** to understand how Python keeps track of **nested function calls**

- 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
- Recognize the different types of **errors** that can be raised when you run Python code

- 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

- **Index** and **slice** into strings to break them up into parts
- Use for loops to loop over strings by **index**

- Translate algorithms from **control flow charts** to Python code
- Use **nesting** of statements to create complex control flow

Unit 2 - Data Structures and Efficiency

- Read and write code using **1D** and **2D lists**
- Use string/list **methods** to call functions directly on values

- Recognize whether two values have the same **reference** in **memory**
- Recognize the difference between **destructive** vs. **non-destructive** functions/operations on **mutable** data types
- 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 **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

- 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 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 **tree**, a **binary tree**, or a **binary search tree (BST)**
- Search for values in trees using **linear search** and in BSTs using **binary search**
- Analyze the **efficiency** of binary search on **balanced** vs. **unbalanced** BSTs
- Recognize the requirements for building a good **hash function** and a good **hashtable** that lead to **constant-time search**

- Identify **brute force approaches** to common problems that run in **O(n!)** or **$O(2^n)$**, including solutions to **Travelling Salesperson**, **puzzle-solving**, **subset sum**, **Boolean satisfiability**, and **exam scheduling**
- Define the complexity classes **P** and **NP** and explain why these classes are important
- Identify whether an algorithm is **tractable** or **intractable**, and whether it is in **P**, **NP**, or **neither** complexity class
- Use **heuristics** to find good-enough solutions to NP problems in polynomial time

Unit 3 - Scaling Up Computing

- Recognize and define the following keywords: **concurrency**, **parallel programming**, **CPU**, **scheduler**, **throughput**, **multitasking**, **multiprocessing**, and **deadlock**
- Calculate the **total steps** and **time steps** taken by a parallel algorithm
- Create **pipelines** to increase the efficiency of repeated operations by splitting steps across cores

- Recognize and define the following keywords: **distributed computing**, **cloud computing**, **browsers**, **routers**, **ISPs**, **IP addresses**, **DNS servers**, **protocols**, and **packets**.
- Use the **MapReduce pattern** to design parallelized algorithms for distributed computing
- Understand at a high level the **internet communication process** that happens when you click on a link to a website in your browser.

- Recognize and define the following keywords: **fault tolerance**, **bottlenecks**, **net neutrality**, **data privacy**, **data security**, **DDOS attacks**, and **man-in-the-middle attacks**
- Recognize and define common approaches of **authentication**, including **passwords** and **certificates**
- Recognize and define the core elements of **encryption**, including **plaintext**, **ciphertext**, **keys**, **encoding**, **decoding**, and **breaking**
- Trace common **encryption** algorithms, such as the **Caesar Cipher** and **RSA**, and recognize whether they are **symmetric** or **asymmetric**

- Read and write data from **files**
- Implement and use **helper functions** in code to break up large problems into solvable subtasks

- Install **external modules** and **import** them into files
- Learn how to use new libraries by using **documentation** and **tutorials**

Unit 4 - CS as a Tool

- Identify whether features in a dataset are **categorical**, **ordinal**, or **numerical**
- Interpret data according to different **protocols**: CSV and JSON
- Use string operations and methods to extract data from **plaintext**
- **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**

- Identify the three major categories of learning (**supervised**, **unsupervised**, and **reinforcement**) and the three major categories of reasoning (**classification**, **regression**, and **clustering**)
- Decide which combination of **learning** and **reasoning** categories are best used to solve a stated problem

- 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

- Update a model after **events** (mouse-based and keyboard-based) based on **rules**
- Use **Monte Carlo methods** to estimate the answer to a question

- Describe how **training**, **validation**, and **testing** are used to build a model and measure its performance
- Recognize how AIs attempt to achieve **goals** by using a **perception**, **reason**, and **action** cycle
- Build **game trees** to represent the possible moves of a game
- Use the **minimax algorithm** to determine an AI's best next move in a game

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, potentially including: **TBD**.