

- 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