Review Notes

Lists and Methods

List (`list`): ordered collection of data values (`[1, 'a']`)
2D List: a list containing other lists

List operations: `+`, `*`, `[]`, `[:]`, `in`

List functions:
`len(L)` - # items in `L`
`min(L)`/`max(L)` - min/max item in `L`
`sum(L)` - sum of items in `L`
`random.choice(L)` - random item in `L`

Additional string functions:
`ord(s)` - ASCII number of `s`
`chr(x)` - ASCII value of int `x`

Method: a function called directly on a data value

```
value.method(args)
```

Methods:
`s.isdigit()`/`s.islower()`/
`s.isupper()` - checks that property of `s`
`L.count(item)` - # times `item` appears
`L.index(x)` - index of `x`, `-1` if missing
`s.lower()`/`s.upper()` - makes new version of `s` that is lowercase/uppercase
`s.replace(a, b)` - new version of `s` with `a` replaced by `b`
`s.strip()` - new version of `s` with extra whitespace removed
`s.split(delim)` - makes a list of parts of `s` separated by `delim`
`delim.join(L)` - makes a string of parts of `L` joined by `delim`

References and Memory

Reference: an address in memory. Connects a variable to its value.
Memory: sequences of bytes where data values are stored.
Aliased: two variables that share the same reference.

`a is b` - is `a` aliased to `b`?

Mutable: a data type that can have data values modified directly in memory, often via methods.
Immutable: a data type that cannot be modified directly in memory; all changes must be made by changing the variable reference.

Destructive: a type of action that updates a data structure by modifying values in memory.
Non-destructive: a type of action that updates a data structure by moving the reference to a new memory location with new data values.

Destructive list methods:
`L.append(val)` - adds `val` to end
`L.insert(pos, val)` - adds `val` into index `pos`
`L.extend(L2)` - adds elements from `L2` to end of `L`
`L.remove(val)` - removes `val` from `L`
`L.pop(pos)` - removes item at index `pos` from `L`
`L.sort()` - sorts `L`
`random.shuffle(L)` - shuffles `L`

Recursion

Recursion: an algorithmic technique where you solve a problem through use of delegation instead of iteration.
Base case: a problem state that is so simple, it can be solved immediately.
Recursive case: a problem state that can be solved by recursively solving a smaller version of the problem, then combining that solution with the leftover part.

```
# Simple Recursion Template
def recursiveFun(problem):
  if ____: # base case
    return ____
  else: # recursive case
    smaller = ____
    result = recursiveFun(smaller)
    return ____
```

Multiple recursive calls: a technique where you call the function multiple times in the recursive case, instead of just once. Fibonacci and Towers of Hanoi use multiple recursive calls to simplify problem solving.

Search Algorithms

Linear Search: a search algorithm where you search a list for an item by checking each item sequentially from left to right.

Binary Search: a search algorithm where you search a sorted list for an item by checking in the middle, then eliminating half the list based on comparison to the target. Repeat until target is found or there is nothing left to search.

Hashed Search: a search algorithm where you search a hashtable for an item by running a hash function on the item, going to the appropriate bucket, and searching that bucket for the item. Leads to very fast search when used with a good hash function and a large-enough hashtable.

Hash function: a function that maps an immutable value to an integer. Must be consistent, and must generally map different values to different results.

Dictionaries

Dictionary (`dict`): collection of key-value pairs of items (`{ "a" : 1, "b" : 2 }`). Dictionaries index on key, not position.

Dictionary Operations:
`d[key]` - evaluates to value paired w/ `key`
`d[key] = value` - add/update pair w/ `key`
`key in d` - check if pair w/ `key` is in `d`

Dictionary Functions/Methods
`len(d)` - # pairs in `d`
`d.keys()` - all keys in `d`
`d.values()` - all values in `d`
`d.pop(key)` - remove pair w/ `key` from `d`

For-iterable loop: a for loop over an iterable value instead of a range.
Iterable: a type of value that can be looped over directly. Often composed of individual parts. Examples: strings, lists, dictionaries

```
for item in iterableValue:
    forBody
```

Review Notes

Runtime and Big-O Notation

Best case: an input that leads to an algorithm taking the least steps possible
Worst case: an input that leads to an algorithm taking the most steps possible

Function family: a set of functions that all grow at a similar rate (eg, linear functions) expressed in a simplified format.
Common function families: constant, logarithmic, linear, quadratic, exponential

Big-O: a representation of the function family of the worst-case scenario for a specific algorithm. Represented as O(runtime).
Common Big-O runtimes: O(1), O(log n), O(n), $O(n^2)$, $O(2^n)$

Linear Search: O(n)
Binary Search: O(log n)
Hashed Search: O(1)

Trees

Tree: a data structure composed of nodes holding values that are connected hierarchically in a recursive manner.
Binary Tree: a tree where each node has at most two children, called left and right

Parent: the node connected directly above the current node
Children: the nodes connected directly below the current node
Root: the topmost node of a tree (with no parent)

Our tree format is a recursively nested dictionary:

```
{ "contents" : nodeValue,
  "left"     : leftChildSubtree,
  "right"    : rightChildSubtree }
```

If there is no left/right child, the key maps to `None` instead.

The common algorithm structure for trees is recursive:
Base case: when the tree is a leaf, or an empty tree
Recursive Case: recursively call the function on the left child and right child (if they exist) and combine the results with the current node

Graphs

Graph: a data structure composed of nodes holding values connected by edges
Neighbors: a pair of nodes connected by an edge.
Directed: a graph where edges can go from one node to another and not vice versa. The opposite is undirected.
Weighted: a graph where edges have values (called weights). The opposite is unweighted.

Our graph format is a dictionary mapping nodes to lists of neighbors:

```
{ nodeValue : [ neighborValue ],
  ... }
```

If the graph is weighted, neighbors are represented as value-weight pairs:

```
{ node : [ [neighborValue, weight] ],
  ... }
```

Object-Oriented Programming

Class: a program code template for creating objects
Writing class: writing constructors that let us pre-load instances with properties, for example,

```
def __init__(dog, name, age):
    dog.name = name
    dog.age = age
```

Advantage of OOP: Encapsulation (organize code and restrict data of an object) and Polymorphism (the same method name can run different code based on type)

Search Algorithms II

Binary search tree (BST): a tree for which the left child of every node (and all its children, etc) are strictly less than the node, and the right child of every node (and its children, etc) are strictly greater than the node.
Binary search: an algorithm that can be performed on a BST by making just one recursive call - to the left if the target is smaller than the root, to the right if larger.
Binary search runtime: binary search on a BST runs in O(log n) if the tree is balanced (all left and right subtree pairs are ~ the same size), O(n) if unbalanced.

Breadth-First Search (BFS): an algorithm where you search for a path from a start node to a target by visiting the immediate neighbors, then their immediate neighbors, etc, until the target is found or no neighbors remain.
Depth-First Search (DFS): an algorithm where you search for a path from a start node to a target by visiting a string of neighbors until you reach a dead end, then backtrack to the most recent unvisited neighbor; repeat until target is found or all neighbors have been visited
BFS and DFS runtimes: the runtimes depend on the number of edges in the graph. If we assume each node has constant # edges, both run in O(n).

Tractability

Brute force approach: an algorithmic strategy - solve a problem by generating all possible solutions and checking them.
Travelling Salesperson: a problem where you find the shortest route across all nodes in a graph. Runs in O(n!).
Puzzle Solving: a problem where you solve a jigsaw puzzle by finding an arrangement of pieces that fits all constraints. Runs in O(n!).
Subset Sum: a problem where you find a subset of numbers in a list that sums to a target number. Runs in $O(2^n)$.
Boolean Satisfiability: a problem where you find a combination of inputs that makes a circuit output 1. Runs in $O(2^n)$.
Exam Scheduling: a problem where you find an arrangement of exams across *k* timeslots such that no student has a conflict. Runs in $O(k^n)$.

Tractable: a problem is tractable if its worst-case runtime can be represented as a polynomial equation. Opposite is intractable.

Complexity class: a collection of function families that have similar efficiency for

certain tasks and are bounded by (no worse than) a certain runtime.

P: a complexity class of problems that are tractable to *solve*
NP: a complexity class of problems that are tractable to *verify*
P vs NP: a big unsolved problem in CS. Are the complexity classes P and NP the same? We don't know!

Heuristic: a search technique used to find good-enough solutions to problems. Generates scores to choose next steps instead of using brute force.

Levels of Concurrency
CPU: a piece of hardware that runs the actions taken by a program

Circuit-level Concurrency: run concurrent actions on a single CPU by running multiple simple circuits at the same time
Multitasking: alternate rapidly between programs on a single CPU to simulate concurrency
Multiprocessing: run multiple programs concurrently on multiple CPUs on the same computer
Distributed Computing: run multiple programs concurrently on multiple computers that are connected together

Concurrency tree: a tree that represents how an expression can be broken down into individual actions, and how those actions can be organized to minimize time spent computing. The leaves are initial values, the root is the final result.

Total steps: the actual number of steps taken in a concurrent process
Time steps: the number of concurrent steps taken by a concurrent process

Parallel Programming
Design difficulty: you have to figure out how to split up steps in a program so they can happen concurrently
Resource sharing: two programs that run concurrently might both want to access the same resource. Resolved using locks.
Deadlock: occurs when 2+ programs are both waiting on resources that one of the other programs in the group already has

Pipelining: make an algorithm concurrent by splitting the process into steps. Each step is assigned to one worker (CPU), data is passed between CPUs.

MapReduce: make an algorithm concurrent by splitting the data into many smaller datasets.
Mapper: part of MapReduce. Takes a small piece of data, processes it, and returns a result.
Reducer: part of MapReduce. Takes a collection of results and processes them into the final result.
Manager: part of MapReduce. The manager moves data through the system and outputs the final result.

How the Internet Works
Browser: program that receives data from the internet and displays it as a webpage
Router: a device that can send data to other machines connected to it via cables. Routers form the core of the internet.

ISP: a service that connects individuals' computers to the internet
IP Address: a series of numbers that forms the 'real name' of a computer. Can be used to find a computer on the internet
DNS Server: a computer on the internet that can map URLs to IP Addresses.
Protocol: a standard for communicating information between machines. Examples include HTML and HTTP.
Packet: a small piece of data that is sent from one computer to another across the internet. Has sender, receiver, and data.
Fault tolerance: the internet is designed to be error-tolerant by being decentralized with many redundancies.

Authentication and Encryption

Data Privacy: we may want to have control over who has access to our data and what others do with it
Data Security: we may want to keep some communications accessible only by the sender and receiver

DDOS: a security attack where a server is overwhelmed by a large number of messages, which blocks regular traffic
Man-in-the-middle: a security attack where a router intercepts data that passes through it to read and/or change it

Authentication: a process to verify someone's identity. Can be done with passwords and certificates.
Encryption: a process to encode data so that only the sender and receiver can read it. Original messages are plaintext; encrypted messages are ciphertext.

Key: a piece of information used to encode a message. Keys can be symmetric (known by both parties) or asymmetric (split into public/private parts)

Encrypt: take plaintext and change it with a key into ciphertext.
Decrypt: take ciphertext and change it back to plaintext using the key.
Break: an attempt to decipher plaintext out of ciphertext without the key

Caesar Cipher: an encryption algorithm where you shift each letter by a set amount. The shift amount is the key.

RSA: an encryption algorithm where you raise the message to a power e and mod by n to encrypt, then raise it to a power d and mod by n to decrypt. Uses asymmetric keys. Public key is (e, n); private key is (d, n).

Keyspace: the number of possible keys that could be used to encrypt a message. Represented as a power of 2 (power is # bits needed to represent key). Caesar Cipher has a keyspace of 2^5; RSA has a keyspace of 2^b.

Managing Large Code Projects

```
f = open(fname, mode) # open file
f.read() # read contents as string
f.readlines() # read list of lines
f.write(text) # write text to file
f.close() # close file when done
```

Helper functions: when given a complicated task, break it into subtasks

and assign each subtask to a separate function to simplify the program.

Learning about Libraries

External library: a library outside of the main Python language that can be installed into Python.
Documentation: instructions on how to use a library available online. Describes existing functions and what they do.

```
pip install name
```

```
import json # JSON library
json.load(f) # read file into data
json.dump(data, f) # write data
```

You can extract data from plaintext using *string operations and methods* like slicing, `split`, `index`, and `strip`.

You can *reformat* data by adding, removing, and reinterpreting existing data using destructive actions like `append`, `pop`, and index assignment.

Data Analysis (I and II)

Data Analysis: gaining insights about data using computation.

Categorical: a type of data that falls into multiple separate categories
Ordinal: a type of data that falls into categories which can be compared
Numerical: a type of data that can be represented by numbers

CSV: data stored in a table-like format, separated by newlines and commas
JSON: data stored in a particular nested format similar to a dictionary
Plaintext: data that does not match a known protocol but can be read directly

```
import csv # CSV library
csv.reader(f) # read f into data
list(reader) # data -> 2D list
csv.writer(f) # write data as CSV
writer.writerows(data)
# actually write the data
```

```
import statistics # stats library
statistics.mean(data) # mean
statistics.median(data) # median
statistics.mode(data) # mode
```

Calculate probabilities over data using `count` and `len`

Visualization: representing data in a visual format. Plot type can be chosen based on the number of dimensions of data (one, two, or three) and the data types being used (categorical, ordinal, numerical).

Visualization options: bar chart, box-and-whiskers plot, bubble plot, colored scatter plot, histogram, pie chart, scatter plot, scatter plot matrix

Matplotlib: a library that enables building visualizations. Plots can be found by searching the APIs and examples.

```
import matplotlib.pyplot as plt
```

```
plt.scatter(x, y) # create plot
plt.show() # render visualization
```

Simulation (I and II)

Simulation: an automated imitation of a real-world event

Model: a computational representation of the real world
Components: information in a model that describes the state of the world
Rules: information in a model that describes how it changes over time or due to events

True randomness: randomly generated numbers that are impossible to predict in a way that allows a winner in the long run
Pseudo-randomness: numbers generated randomly by an algorithm. Can be predicted if you know the algorithm.

Monte Carlo method: solve a problem by running a simulation many many times and averaging the results

```
# Monte Carlo structure
def getExpectedValue(numTrials):
    count = 0
    for trial in range(numTrials):
        result = runTrial()
        if result == True:
            count = count + 1
    return count / numTrials
```

Machine Learning

Machine Learning: algorithmically find patterns in data and automatically develop a model for the data based on them.

Supervised learning: learn from *labeled* data to predict label outputs based on the given information. Can be used to make predictions on future data.
Unsupervised learning: group *unlabeled* data into categories by finding data points that are similar to each other. Helps find natural structures, but hard to test.
Reinforcement learning: help an *AI agent* solve a goal by repeatedly checking whether the agent is closer too or further from the goal.

Classification: given labeled data, produce a model that can find *categorical or ordinal* results.
Regression: given labeled data, produce a model that can find *numerical* results.
Clustering: given unlabeled data, group similar data points into separate *clusters*.

Training: use most of the original dataset to look for groups of features that correctly predict results. Build these features in a model to make future predictions.
Validation: while training, repeatedly test on a subset of the data to make sure the model isn't overfitted to the known data
Testing: when the model is finished, test it once on a reserved subset of the data to see how well it performs. Since the model has never seen this data before, it should perform similarly on new, unlabeled data.

Artificial Intelligence

Review Notes

Artificial Intelligence (AI): computational techniques that attempt to mimic signs up human intelligence using programming

Agent: a model trained by an AI algorithm to accomplish a specific task. Works through perception, reason, action cycle.

Perception: gather information about the problem being solved
Reason: given the current information, decide what should be done next, often using search algorithms
Action: take the chosen action to move closer to the goal

Game Tree: a tree that represents all the possible states of a two-player game. Nodes are game states (possible board configurations), edges are actions taken by players.

Minimax: an algorithm that can be applied to a game tree to help the AI choose the best possible move. Scores leaves based on whether the AI wins (1) or loses (-1); for inner nodes, chooses the maximum child score if the AI is taking the turn, or the minimum score is the user is taking the turn.

Heuristic: heuristics can be applied to game trees by only moving down a certain number of levels, then scoring inner nodes with a heuristic function instead of creating the full game tree.