15-110 Quiz3 Notes Sheet

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`

15-110 Quiz3 Notes Sheet

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
```