**Typos:**

* Returning a void function instead of res
* Forgetting to increment I in a while loop
* Forgetting to set n = len(s[0]) – 1 instead of len(s[0]), causing index out of range
* For x in range(1,len(nums) needs to be range(i+1,len(nums))
* Range(n+1), not range(n)+1
  + Make sure you loop through the whole DP array (off-by-one)
* Use for \_ in range(x) if you don’t need the index
* For key,value in dict.items() is the syntax for getting all key, value pairs
* Second argument to filter is iterable, first argument is the function
* Syntax for creating a new thread: Thread(**target**=get\_nbors,**args**=(url))
* I < 6 NOT I < len(6)
* Python doesn’t include a builtin nCr function
* Use set.update() to add elements to sets

**Mistakes:**

* Using a DFS/backtracking technique where an iterative DP array technique is required
* Forgot to account for the case where the words have different length
* Using .add() to add to python Counter(), need counter[elem] += 1, since conter extends dict
* Counter.most\_common(), NOT Counter.most\_freq()
* Forgetting to increment i/j in while loops
* When using list.sort(lambda x: compareFxn), the compare function says which values to put first, not last
* Use functools.cmp\_to\_key() to write custom sorting functions
  + List.sort(key = cmp\_to\_key(compareFn))
  + Need to return <0, 0, >0 rather than True, False in custom compare methods
  + Can also make a custom key function that returns (-1,0,1) in this same pattern
* Num = random.sample(S : set, k : int) to get randoms from set
* Num = random.choice(L : list) to get a random element from a list
* Not accounting for duplicate inputs (whether it be into lists, sets, or dictionaries)
* Not having partition memorized
* Index out of range on random.randint(0,k) needs to be k-1
* Forgetting to write self. In python classes
* Need to write list(dict.keys()) since keys() is a view object not a list
* Need to use iterables to edit vars inside of nested functions in Python
* If you want to persist shared memory while doing a BFS/DFS, do so iteratively
* Forgetting to increment DP[i] by one after looking at the previous DP array
  + Inf is not normally defined, need to use float(“inf”)
  + Make sure inf is a string
* Keep forgetting to increment I in while loops
* Also, max(iterable, **KEY =** lambda x : …)
* Dict.items() returns a list of tuples, whereas dict.keys(), dict.values() returns a view object that must be turned into a list()
* Heapq.heapify() returns None
* Can use self.var to make variables global
* Increment Freq\_dict before changing any indices!

**Conceptual:**

* Using a hash set only has O(1) average case for item retrieval and has O(n) worst case
  + Therefore, lookup from this set is not garunteed O(1)
  + This also depends on the implementation, since sets can also be *balanced* BSTs (AVL, Red-Black) which have O(logn) average case and worst case lookup times
  + In CPython, sets are implemented as dicts with dummy values
  + In CPython, dicts are implemented as hash tables not BSTs
* For longer questions like Solve\_Sudoku keep answers brief so that they are timely to write in an interview
  + Don’t try to implement perfect data structures if it will take too long
  + Same idea for problems that can be solved with Djikstra’s, sometimes it’s not worth it to write that all out
* Look alive when doing any kind of coding assessment, I did one on LinkedIn today and really didn’t try hard enough
  + Also, I don’t really know Java…
* When returning from tree-traversal, return the nodes not their coordinates
* Should account for the possibility of Nbor being None when traversing a tree as a graph.
* When looking at a problem that has a lot of “bucket” intervals, just update the endpoints of those intervals to reduce the runtime from O(n^2) to O(n) in many cases
* When keeping track of occurrences of letters, numbers in lists use counts rather than lists of indices wherever possible!!!
* Finding the median of a given set of numbers can be done when the nums are sorted from least to greatest (odd = middle element, even = avg. of 2 middle elements)
* Can use 2 heaps in findRunningMedian problems
* **Modularize code before writing specific helper functions (this makes things a lot easier!)**
* I forgot how to write the merge function for mergeSort ;(
* Use string-builder arrays to build up string results, however it is no more efficient to hash these (as tuples) than simply hashing strings
* Python string indexing can’t index out of range
* DP tables often start with the empty string
* **If only using adjacent squares in a DP matrix, you can cut down the memory consumption of the problem by only keeping adjacent rows in memory!**
* Should use bitmaps instead of sets to keep track of visited for DP/graph problems where possible, this vastly improves space consumption

**Interview Mistakes:**

* Just did my first HackerRank for practice
  + I got way too nervous because of caffeine consumption, other than that the questions were pretty easy and I could have solved them if I was in a better state of mind
* Need to stay calm and focused during the whole ~80 minutes, write the same clean code you would on a LeetCode problem
* Simply, I need more practice in a **timed** format to do better on these tests
* Use itertools to generate combinations where possible to save time.
* Generally, taking too long on easy problems
  + This will improve with practice, however I need to be very methodical about these or I won’t get it
* Running out of time on interviews so that I’m too flustered to actually complete medium and hard interview problems
  + I had the right first idea (sorting) for merge intervals, but I wasn’t in the thinking state to get it done right then…