**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

**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

**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