**Tech Interview Knowledge**

# |\*:\*||\*:\*|What to Expect|\*:\*| |\*:\*|

# The above is the sample topic

## But here is the subtopic

### And the subtopic below that

# For the phone interview

## First 10-15 minutes are good to know if they’re handwaving or not

# Important topics for the 10-15 minutes that demonstrate ability

## What Projects have you done?

### How deep can you go into the technical knowledge of that project

### What is your communication

### What is your train of thought

### Interest level

# Beyond Leetcode

## Question will dive deeper

## Multiple followups

### Be able to follow-up and build-up on the questions they ask

# |\*:\*||\*:\*|5 Axes for Technical Interviews|\*:\*||\*:\*|

# Coding

## Quality of your code

## If it looks good

## Structures in place

## Well Isolated

# Data Structures and Algorithms

## Reverse linked lists

## Put in proper data structures

## Timespace analysis

# Design (Senior)

## Your analysis of the problem

### Your Big O analysis

### Provide alternatives

# Communication

## Communicate WHILE YOU WHITEBOARD

## Be able to talk fluidly

## Be able to take feedback

## Technical clarity of communication

### Not too high level

### Without getting too deep in the weeds

# Efficiency

## Must be able to work quickly

## A good candidate will be able to get through all questions

# |\*:\*||\*:\*|Behind the Scenes of a Technical Interview|\*:\*||\*:\*|

# No set questions

# Except Dynamic Programming questions

## (you get or you don’t lmao) at Facebook

# Sometimes they don’t know what to ask

# You’re trying to get a signal for those questions

# Roles are generalists at BIG COMPANIES

## Ability to learn new things

## Learn new technologies and tech stacks that are proprietary

# Startups are more specific

# Test of

## General Intelligence

## Enthusiasm

# Don’t be a Jerk

## Be friendly

# DO NOT BE STUBBORN

## Don’t defend the answer

## Communicate with the feedback

# General cool traits

## Eagerness to learn

## Friendly collaboration

## Technical curiosity

## Learning from mistakes

## Willingness to accept feedback and improve oneself

## Get-it-done attitude

## Results-oriented

# |\*:\*||\*:\*|Path to the Onsite (Preparing your resume)|\*:\*||\*:\*|

# Can I get a referral would be nice

# The better you know someone the easier it may be for them to give you more credit

# Message associated recruiters on LinkedIn

## Cover Letter ready

### Explain why you are qualified

# Startups have an even higher referral requirement

# For your projects (see my work)

## Talk about it

## Understand how it was made

## Or why even

# Focus on the product and its impact

## Your deliverable and its necessities

## |\*:\*|YOU NEED TO WRITE OUT YOUR SCRIPT|\*:\*|

### Aka whats on your resume, but the elevator pitch version of it

# Have someone look at your resume

## They can find the problems there

## Highlight the technologies that you used

# |\*:\*||\*:\*|How to apply for interviews|\*:\*||\*:\*|

# Apply to companies you don’t CARE THAT MUCH FIRST

# Rank your companies

## A,B,C

## Go for C/B first

## Can be A

# Sometimes it can be a bluff if they set a timeline

## Only decline if you have an offer in hand

# |\*:\*|WHITEBOARDING!!!!!!!|\*:\*|

## Holy shit you have to practice by hand

## Give yourself 10-15 minutes

## 98% of your time is leetcode

### Basically like studying for an exam

# Blameless culture, ability to learn from mistakes

## |\*:\*|WHAT ARE YOUR EXAMPLES OF LEARNING FROM MISTAKES|\*:\*|

### The fault is from the process, where did it go wrong

## What was the issue?

### How did you learn from that?

## Foundations of becoming a senior engineer

### DO NOT PRETEND TO KNOW EVERYTHING

# What is your passion: BUILDING!!

## Remember the hackathon

# Senior Positions

## System Design questions

# |\*:\*||\*:\*|Axis 1: Coding|\*:\*||\*:\*|

# L3 engineer is basic experience (easiest to go)

## L4 is even better

## L5 seems like a senior and really experienced (that’s 5 years though)

# Prepare for a specific set

## FE/BE

# Show your cross-functional skills

## Leadership

## UX Communication

## MySQL

## Database

# Coding Characteristics

## Can you program

## Is it production quality

## Production quality

# I.E. OOP: Is Palindrome (Recursion)

## |\*:\*|Interviewers won’t write the method function signature|\*:\*|

### They expect you to see what is MISSING FIRST

## Wrong approach

### Start working right away

### Adding too many parameters

### Wrong return value

### Unclear helper functions

## Right approach

### DID YOU WRAP IT UP IN A CLASS

### Easy to read input/output?

#### Good private/public variables

### Clean public function

### Private helper function

#### Can have 5 params

#### Manages state

### THIS KEEPS STRUCTURE CLEAN

### Are you using |\*:\*|INSTANCE VARIABLES|\*:\*| for recursion?

#### They help store state

#### A field declared without using the word STATIC

#### The T is, it’s missing in INSTANCE to make STATIC

### GLOBAL VARIABLES ARE OK IF THEY ARE CONTAINED IN A CLASS

#### Normally you wouldn’t though

# Make sure your function names are right

## |\*:\*|USE ABBREVIATIONS after a CERTAIN POINT|\*:\*|

### If you abbreviate, explain them!

## Don’t write really long variable names after they get the idea

# Typed languages (i.e. Java) can lose you speed

## Untyped like Python will be faster….

## |\*:\*|LEARN THE FASTEST WAY TO WHITEBOARD JAVA|\*:\*|

## I.E. How do you write a nested Hash Table/Dictionary

### In Java you have to explicitly explain

### In Python you do not

# |\*:\*|Be able to write a recursive function|\*:\*|

## Practice writing a lot

## You have a helper function, but you don’t see it

## Sometimes an iterative is better (if not most of the time)

## Recursive is limited to a stack’s space

## If it’s Recursive your space complexity is never O(1)

### Because you always need to store the state

## Functional programming can use recursion more

## |\*:\*|I.E. Traverse a tree in order|\*:\*|

### Recursion is best for that

### BUT WHAT IF THEY ASK YOU TO IMPLEMENT IT ITERATIVELY

#### You have to demonstrate knowledge of EXACTLY HOW RECURSION WORK

### |\*:\*|Iterative Trick (USE STACKS)|\*:\*|

#### Put states in stack

#### Pop out the state when you’re done

### Overall, understand the FUNDAMENTALS first

# |\*:\*|Be able to convert a recursive function to an iterative function (Practice)|\*:\*|

## Instead of using recursion, use a |\*:\*|STACK|\*:\*|

## Iterate over it, use a |\*:\*|WHILE LOOP INSTEAD|\*:\*|

## You have a stack, push the first item/state on to it

## Then start while loop

### Each Iteration will pop the item off the stack

### Process it

### If you have more objects, then PUSH THOSE ON

### May need a for loop

#### Iterate on the new items

## END when no more items are in the stack

# Do not use only pseudo code – WRITE REAL CODE

## Must be something that can compile

# |\*:\*|Reverse a string using recursion|\*:\*|

## class Solution:

## def reverse(self, string):

## if len(string) == 0:

## return string

## else:

## return self.reverse(string[1:]) + string[0]

## def reverseIterative(self, string):

## answer = ''

## stack = [string]

## while len(stack):

## item = stack.pop()

## answer += item[-1]

## nextItem = item[:-1]

## if len(nextItem):

## stack.append(nextItem)

## return answer

## a = 'hello'

## print Solution().reverseIterative(a)

### This one is a little weird because in the first recursive example, he builds it right to left

### But in the iterative it builds the string left to right

### Better to keep it consistent where possible

# |\*:\*||\*:\*|Axis 2: Data Structures and Algorithms|\*:\*||\*:\*|

# Knowing the right data structure to use will be the key to most optimal solutions

## Impossible to solve some of these questions

# The hardest part of solving: Translate the problem into a proper data structure/algorithm

# List of structures to know

## Arrays

### Some sets

## Hash Maps/Dictionaries/Associate Arrays

### MOST IMPORTANT

## Queue/Stack

### Used all over in leet code

### |\*:\*|SEE BFS/DFS|\*:\*|

### Seen a lot in graphs and trees

## |\*:\*|Trie (given a dictionary come up with auto completion)|\*:\*|

### Strings, Super important in string algorithm

### Each node represents a letter

#### First node is an empty string

#### Every node after that can be described more letters to form strings

### Common usage to find how many words start with “ASB” or whatever combination

### A delimiter may be used to denote the very end of a trie

## 

## Tree Traversal

### ALWAYS HERE

## Common to see data structures combined

### An hash map of arrays for example

## UNCOMMON

### Djistrka algorithm is too long

### Less priority queue, heaps, stacks

# If something is taking too long, you’re probably taking the wrong approach

# I.E. Implementing a grid

## You could use a 2d array

### But you would have a lot of difficulty implementing bound checks

# |\*:\*|Proper use of data structures is KEY, because SPEED IS KEY|\*:\*|

## 90% of candidates will get the problem right

## But won’t be able to get it done quickly

# Create a robust solution that saves time (most important)

# Could you get an NP complete problem?

## Uh… Maybe

## |\*:\*|Look into this I guess|\*:\*|

# Sample progression of problem difficulty

## Demonstrate the problem recursively

## Now do it iteratively (while + stack)

## Now do it assuming you have multiple processors on hand

### |\*:\*|WTF DO I DO NOW|\*:\*|

## Now do it with no space, BIG O

### Optimization

# Know your libraries very well, ask your interviewer if you are allowed

## Queues

## Stacks

## Heaps

# You should have an intuition of what the problem is to solve at this point

## All are very cookie cutter

## Here’s a tree

### Clone that tree

## Here’s a string

### Reverse it

### Then look for the duplicate letters

# The hardest part is GETTING STARTED

## It’s hard at first

## After the ramp up time, the patterns are so obvious

## I.E. A person is friends with another person…

### GRAPH DATA STRUCTURE immediately

# |\*:\*||\*:\*|Axis 3: Design|\*:\*||\*:\*|

# Design is alternatives, Big O timespace analysis

## A lot of the time can you guess it

## |\*:\*|But did you have the ability to analyze it efficiently|\*:\*|

# |\*:\*|Can you in fact give the timespace analysis first|\*:\*|?

## I.E. if we do it this way, it will be linear time, linear space

## Even go down to alternatives

# Google loooves design docs

## Even before something is approved, you will then discuss the alternatives

## |\*:\*|There is a trade off between time and space|\*:\*|

### Hash Map = Linear time algorithm, easily

### Array = Takes longer, but less space requirement

#### Good if you don’t have much memory available

# I.E. If we use…

## A hash map

### We’ll get linear time

## If we don’t use the hash map

### In a brute force solution, we get polynomial or exponential time

# Shows you have good design skills

# I.E. Design a data structure where you can insert an item and then retrieve it

## O(1) to insert

## O(n) to retrieve

# I.E. Twist: What if you have to retrieve a lot of elements?

# I.E. Twist: I want to retrieve FAST, but push SLOW

# I.E. Twist: What if K is large, n is small

## Or vice versa

## Heaps?

# |\*:\*|Process, explain what you would do and time complexity|\*:\*|

## If the interviewer does not react, you could POTENTIALLY do it more optimally

## Pause and look if they’re satisfied

## If so, they might say “Ok, can you code this for me”

# DO NOT JUMP INTO CODING RIGHT AWAY

## With the most brute force solution

# DO NOT keep asking

## Is this ok?

## What do you think?

## You’re supposed to be the one with confidence

# Instead, propose a few designs and check and get the interviewer’s approval

## This helps establish confidence in your solution

## It’s like saying here are options A, B, C

### Here are the timespace requirements

## Be ready to code any of the suggestions you offer

# |\*:\*|Sometimes an interviewer will ask an AMBIGUOUS question|\*:\*|

## The intention is to |\*:\*|GET YOU TO ASK TO CLARIFY|\*:\*|

## To find the requirements

# |\*:\*|Do you want more|\*:\*|

## Time?

## Or Space?

## Which are you optimizing here (pay attention to timespace caveats)

### Is it going to be across a million different entries, or is it super small

### Are you getting a lot of colors, or a few

# Still keep your time management on track

# |\*:\*||\*:\*| Axis 4: Communication and Comprehension|\*:\*||\*:\*|

# Even if you solved the question optimally, you MUST COMMUNICATE it

## If your teammates don’t understand what you’re thinking, it can be very confusing

## If you can’t share your knowledge

### You can’t collaborate -> you can’t build something big

# |\*:\*|The ability to clearly explain what you’ve done|\*:\*|

## |\*:\*|What did YOU individually do|\*:\*|

### Not what your company does

### Not your product

## If there was something technically challenging

### Explain the design of that system to me

### |\*:\*|GO BACK AND REREAD ALL YOUR REWORKS|\*:\*|

# Storytelling

## Your interviewer does NOT have context, so you have to give it to them

## BEFORE you start diving into technical details

## What would make them understand exactly what you did

## And the impact

# Interviewer wants to know what is interesting

## Steer it to that

## Explain your thoughts/algorithm

# |\*:\*|Talking while Whiteboarding|\*:\*|

## You DON’T have to talk continuously

## Instead, Announce what your intention is (i.e. I’m going to write a for loop)

# When you get good, your communication should take MORE time than the actual coding itself

# |\*:\*|What happens if you get stuck?|\*:\*|

## Start explaining the ideas you may have

## And the issues

## |\*:\*|Lets say you are at a brute force approach|\*:\*|

### Stop

### Talk through a few ideas with the interviewer

### Oftentimes, they will not say anything…

### But the problem is not impossible

## Try not to use the interviewer as a crutch

### Aka get good… this isn’t that helpful

## But if they do give you a hint, TAKE IT, do NOT DISMISS IT

### That’s about being able to take in feedback

### |\*:\*|Be able to take criticism, and integrate it to improve yourself|\*:\*|

# |\*:\*|Do not talk TOO LONG, keep it to 5 minutes|\*:\*|

## Your setup may be too long

# |\*:\*||\*:\*| Axis 5: Other Criteria|\*:\*||\*:\*|

# |\*:\*|Efficacy AKA SPEED|\*:\*|

## Don’t pretend to struggle lmao

## They have more questions for you

# YOU NEED TO PREPARE

# TAKE AS FEW BREAKS AS POSSIBLE

## Gives you more time to talk about yourself, solve more problems

# |\*:\*|System Design|\*:\*|

## Given more towards senior candidates

## For the domain space you lean towards

### AKA Data Infrastructure Engineer

### Understand flow of data

### Structure a program correctly

## The biggest idea is what project have you taken from START to FINISH

### What were the results

## How do you

### Get shit done

### Communicate

### Unblock yourself

# After you interview, you’ll be given a score

## You need a person that you are strongly endorsed

## Then you go to hiring manager/committee

## They really are an objective 3rd party to see where you rank

## The interviewers goal is to make a WELL CALIBRATED interview (consistency across all panelists)

# |\*:\*|Skipping the Interviewing Masterclass for now|\*:\*|: <https://www.techseries.dev/products/tech-interview-pro/categories/1408704/posts/4724482>

# |\*:\*|Coding deep dive|\*:\*|

# Is it structured, is it readable

# I.E. Given a grid

## With multiple colors filled in, what is the maximum number of connected colors

# Clarifying questions

## How many different colors are there?

### If there are TOO MANY POSSIBLE COLORS

#### |\*:\*|RECURSION FAILS DUE TO LIMIT ON STACK SPACE|\*:\*|

## With too many colors, an interviewer might expect you to switch from recursion to iterative solution

# Initial state: given a method signature

## |\*:\*|PRACTICE WRITING METHOD SIGNATURES FROM SCRATCH|\*:\*|

## Then wrap it in a class

## User helper methods

# Fibonacci sequence reversed

## <https://www.techseries.dev/products/tech-interview-pro/categories/1408103/posts/4821201>

## Keep in mind Fibbonacci can be solved using memoization (cached)

### I.E. Fib(3) is used in both fib(5) and fib(4)

#### When we encounter it, if it is not stored in a cached location yet, we should place it in a cache

## Keep in note that you want to keep things not exposed for getting the result should be private

### In the case of the grid sample, you keep the traverse function private

## If you have something that has a lot of boilerplate methods

### You can say lets abstract “get neighbors”

## You want to prove to the interviewer that you know what you’re doing

### You want to get semicolons in

### Don’t use too much pseudo code

# Map, flatmap, don’t use too many builtins

## Can still show off, but know the timespace of it

# |\*.\*|Don’t use too many sort functions|\*.\*|

## It’s implied that you should write them

## You won’t know the Big O timeplace

# |\*.\*|Common Recursion -> Iterative|\*.\*|

## Change the name of the method to imply that it is recursive, change previous calls as such

## Think about initial base case

### Initialize the stack

### Append the first state on to the stack

## Pop off the first state

## Apply changes/checks

### Is valid color, sum++

## Prepare for next case, append the next states

## While Loop restarts

# Practice and TIME yourself on both

## Computer

## White board

# General tips

## Start in the top left corner

## Leave space inbetween code

### Can add in when necessary

## Know shorthand where possible

## Use “ “ sometimes

# Style editor

## Java

<https://google.github.io/styleguide/javaguide.html>

## Python

<https://google.github.io/styleguide/pyguide.html>

# |\*:\*||\*:\*|Trees & Graphs Overall|\*:\*||\*:\*|

# <https://www.techseries.dev/products/tech-interview-pro/categories/1408103/posts/4821207>

# Graph

## is a node, connected to many other nodes

## No cycle

# Trees

## Graph with more constrictions

## 0,1,2 nodes

## Never loops back on itself

# |\*:\*|BIG O of trees|\*:\*|

## It’s oftentimes logarithmic

## Based on height of the tree

## I.E. it fans out twice at each node, it’s Log 2

## HOWEVER if you’re just searching for a node

### It’s O(n)

## Big O worst case, what if you have a long line of imbalanced values on the one side?

### It’s a branch

### Then it’s O(n)

# Preorder vs. InOrder vs. Postorder

## There is a specific order to each

## 

## The above is cool and all, but they’ll usually just ask you to do an inorder traversal

## |\*:\*|largest use case is when PRINTING something|\*:\*|

# More importantly, you will see a lot of BFS/DFS

## BFS = ABCDEFG

## DFS = going to a leaf node, always left or right

### Aka anything above

# If you do an inorder traversal of a binary search tree

# Implementation of a tree

## Left vs. Right node

## Children

### Self.data

### Self.children, where children is an array

# If you were ever asked to do an iterative version

## DFS is a stack

## BFS is a queue

# 2 Common tree problems

## Views/Subviews

## You need to print out each view

# In python as in Java, strings are immutable

## You are being provided a new string every time

# Quick Python self-notes

## No type

## No ;

## When you cast you wrap around the input

## Where string1 = input

## Length of string

### Len(string)

## Printing a line

### Print(string)

## String concat

### Print(string1,string2)

### Set ,END if you want a concat OTHER THAN SPACE

## Substring

### String1[2:4]

#### If you leave out the second as a space, it goes to the end

##### This is used in reverse string, sometime you just pass word[1:] as the next recurvis input

## Cool trick, if you want the last 3 letters

### String1[-3:]

## String contains

### (“pu” in s)

## List

### Adding = list.append(“house”)

### Length = len(list)

### Element at index = list[1]

#### Can also access all array elements list[1:3]

### Last element of list = list[-1]

### Insert at index = list.insert(1, “mouse”)

### Remove at index = del(list[1])

#### Can even use del(list[1:3])

## Tuples

### Are immutable

### If x is a tuple, x[0] = whatever it was originally

### If you want to define a tuple via an integer

#### X = 5,

#### The comma turns it into a tuple

### Tuples can use

#### In

#### Len

## COMPARISONS

### ==

#### Checks the values

### Is

#### Check to see if they are the same object

### !

#### Is literally not

## Conditional operators

### Not and or

## Control statements

### NO PARENTHESES

### If, then tab

## Python does not have increment or decrement operators

### x++ = x+=1

## Range increments by 1

### For x in range(5):

#### DON’T FORGET THE SEMICOLON

#### It does NOT include the last value

##### I.E. if the array had values from 0-5, it would print 0-4 not including 5

### For x in range (1,3)

#### 1,2

### For x in range (1,6,2)

#### 1,3,5

## FOREACH and if

### List = [1,2,3,4]

### For x in list:

#### If x ==3:

##### Continue/continue

#### Print(x, end = “ “)

## A for loop can be associated with an else statement

### For x in range(5):

#### If x==3

##### Break

### Else:

#### Print(‘I didn’t break’)

### If the code never entered the break statement, I didn’t break would print

#### It’s associated with the tab

## Functions (no typing)

### Def sum (a, b, c):

#### Return a + b+ c

### Print sum(1,2,3)

## Can even do easy things like associate another variable as a method

### Mystery = sum

### Print(mystery(1,2,3))

#### Returns sum(1,2,3)

## Python naturally supports optionals, no need to overload

### Def sum (a, b, c=4):

#### Return a + b+ c

### Print sum(1,2)

### Your end result is 7

## Common idea of a private variable is

### \_ before the variable name

## Boilerplate Methods in python

### Class Dog:

#### Def \_init\_(self, name, age):

##### Self.\_name = name

##### Self\_age = age

#### Def get\_age(self):

##### Return self.\_age

#### Def get\_name(self):

##### Return self.\_name

#### Def set\_name(self,name):

##### Self.\_name = name

#### Def set\_age(self.age):

##### Self.\_age = age

#### Def \_\_str\_\_(self):

##### Return “Dog:\nName: “ + self.\_Name + “\nAge: “ + str(self.age)

#### Def bonus\_method():

##### Return “hello”

### D1 = Dog(“Scruffy, 5)

### Print d1

### NO NEW OPERATORS, just assign

### If you wanted to call a class method in dog (i.e. the bonus method at the bottom)

#### Print(Dog.bonus\_method())

##### “hello”

# |\*.\*|Sliding window?|\*.\*|

# |\*:\*||\*:\*|Coding Problem|\*:\*| Number of Islands|\*:\*||\*:\*|

## <https://www.techseries.dev/products/tech-interview-pro/categories/1408104/posts/4821250>

## <https://leetcode.com/problems/number-of-islands/>

## Difficulty: Medium

## Keywords: Graph, DFS, BFS

## Big O

### Space of (m\*n) in worst case where entire grid is filled with 1’s

## Tricks/Takeaways

### The idea of a DFS

## A screenshot of a cell phone Description automatically generated

## #Big O space of O(m\*n), in worst case where the entire grid is filled with 1's

## class Solution:

## def numIslands(self, grid: List[List[str]]) -> int:

## counter = 0

## #sink island called recursively on each valid point

## def sinkIsland(r,c,grid):

## if grid[r][c] == '1':

## grid[r][c] = '0'

## else:

## return

## if r+1 < len(grid):

## sinkIsland(r+1,c,grid)

## if r-1 >= 0:

## sinkIsland(r-1,c,grid)

## if c+1 < len(grid[0]):

## sinkIsland(r,c+1,grid)

## if c-1 >= 0:

## sinkIsland(r,c-1,grid)

## 

## #start in upper left corner [0][0]

## #iterate when something new is '1'

## for r in range(len(grid)):

## for c in range(len(grid[0])):

## if grid[r][c] == '1':

## counter+=1

## sinkIsland(r,c,grid)

## return counter

# |\*:\*||\*:\*|Coding Problem|\*:\*| Perfect Number|\*:\*||\*:\*|

## <https://www.techseries.dev/products/tech-interview-pro/categories/1408104/posts/4821251>

## <https://leetcode.com/problems/perfect-number/>

## Difficulty: Easy

## Keywords: No actual data algorithm, uses math and reduce imports

## Tricks/Takeaways

### You forgot the base case

## Big O

## import math

## from functools import reduce

## class Solution:

## def checkPerfectNumber(self, num: int) -> bool:

## factors = []

## sum = 0

## #in case there is a negative number

## if num <= 0:

## return False

## #round up, so like in 50\*\*.5 = 7.07, end result is 8

## #concept is that the root is the absolute max of the factors

## for i in range(1,int(math.ceil(num\*\*.5))):

## if num % i == 0:

## factors.append(i)

## factors.append(num/i)

## #check to see if factors has values

## if factors:

## #remember to remove the number itself from the perfect number

## sum = reduce(lambda x,y: x+y, factors) - num

## return sum == num

# |\*:\*||\*:\*|Coding Problem|\*:\*| Top K Frequent Elements|\*:\*||\*:\*|

## <https://www.techseries.dev/products/tech-interview-pro/categories/1408104/posts/4821251>

## <https://leetcode.com/problems/top-k-frequent-elements/>

## Difficulty: Medium

## Keywords: Heaps

## Tricks/Takeaways

### FILL IT OUT

## Big O

# |\*.\*|Invert a binary tree|\*.\*|

# Additional resources

## Cracking the Coding Interview

## Andrei’s Data structures and Algorthims on Udemy

## Leetcode obviously

## <https://leetcode.com/discuss/career/216554/from-0-to-clearing-uberappleamazonlinkedingoogle>

## Elements of Programming Interview

## <https://www.reddit.com/r/learnpython/comments/e9salh/how_useful_is_cracking_the_coding_interview_book/>

## <https://github.com/sh0sh0/Elements-Of-Programming-Interviews>

## <https://github.com/dxmahata/elements-of-programming-interviews-python-solutions>

## <https://github.com/w-hat/ctci-solutions>

# Motivation

## Don't be demotivated at your current level. I started from a place where I was bombing every interview and now I've done much better than what I was expecting. Start and be consistent.

## Try getting comfortable in every topic. Don't leave anything behind.

## Don't spend more than an hour on any question. If you can't figure out the solution, mark it and revisit later.

## Use the discuss forums. They are literally the best part about Leetcode and a key differentiator why no other platform can match up to Leetcode in terms of interview prep.

## Upsolve when you can't solve a problem - i.e look at the solution, understand and then do it again on your own.

## Keep taking notes about what a problem is teaching you. Keep revisiting them. Spaced repitition helps in committing things to memory.

## Don't be in a hurry. Enjoy the ride :)

# |\*.\*||\*.\*|<https://realpython.com/python-coding-interview-tips/#use-list-comprehensions-instead-of-map-and-filter>|\*.\*||\*.\*|

# Use enumerate() to iterate over both indices and values

## Instead of

### for i in range(len(numbers)):

### numbers[i] % 3 == 0

## Use

### for i, num in enumerate(numbers):

### num % 3 == 0

### num is only the value, you still need to set the correct value in an array using

#### numbers[i]

## Also

### Default start is at 0, but you can start at any value

### for i, num in enumerate(numbers, start=52)

# BTW, did you know that this is a valid is\_odd function, because non-zero values resolve true with bool()

## def is\_odd(x):

## return bool(x % 2)))

### I.E. 5 % 2 = 1, ret true

# Use List Comprehensions Instead of map() and filter()

## Instead of doing

### list(map(square, numbers))

#### where square is a function to be applied to the numbers array, and subsequently turned into a list

### OR

### list(filter(is\_odd, numbers))

#### where is\_odd is a Boolean function applied to the numbers array

## Do this

### [square(x) for x in numbers]

### OR

### [x for x in numbers if is\_odd(x)]

#### So if you compare it to the top, you could apply a mapped function if you want, but you just want to read x

#### The if statement helps with filtering

# Debug problematic code with breakpoint()

## Instead of using print(“log val”)

### Use breakpoint()

# Format strings effectively with f-strings

## I hiiiiighly doubt I’m ever going to do this

# Sort lists with custom arguments

## Via sorted()

## By default it’s ascending order

### 1,2,3 or a, b, c

## To reverse

### Sorted([1,2,3], reverse=True)

#### 3,2,1

## Given an array like this, sort by key

### animals = [

### {'type': 'penguin', 'name': 'Stephanie', 'age': 8},

### {'type': 'elephant', 'name': 'Devon', 'age': 3},

### {'type': 'puma', 'name': 'Moe', 'age': 5},

### ]

## You can do that by doing

### Sorted(animals, key=lambda animal: animal[‘age’])

### Will go ele,puma,peng

# Store unique values with SETS

## Predefine and use the SET data structure

### Set()

## Adding a unique element in a set

### setElement.add(random\_string())

## Adding a unique element in a list

### BIG O (N\*\*2)

### word = random\_string()

### If word not in setElement

#### setElement.append(word)

# Interesting for notation to iterate through all

## For \_ in range(1000)

### Print(“hi”)

## You do this because \_ indicates you don’t actually care about the value

# Use generators instead of list comprehensions to conserve memory

## Allows for scalable calculations

## Given an ask, find the sum of the first 1000 perfect squares

## You might create something that uses list comprehension

### Sum([I\*I for I in range(1,1001)])

## The problem with the above is that you are making a list and taking a lot of memory space, what happens when you want to find the first MILLION perfect squares

## Instead you do

### Sum((i\*I for I in range(1,1001)))

### List bracket swapped out for parentheses

## This is a generator object instead, and it only knows the current state

### It calculates i\*I, increments I, and returns the value to sum

### No space taken creating the list item

# Define default values when looking up dictionary keys

## Let’s say normally you will check if a dictionary (Python hashtable) has a value, if it doesn’t have one, set it to its default, otherwise get the value

## Here’s how you do that

## Given dictionary cowboy

### Name = cowboy.setDefault(“name”, “This one has NO NAME”)

# Handle Missing Dictionary Keys With collections.defaultdict()

# Count hashable objects with the collections.Counter class

# Access Common String Groups With string Constants

# Use the standard library to get lists of permutations and combinations

# Yield vs. Return

## Yield gives back a value, but does not end or forget the state and resumes right after displaying

## Return forgets the state and starts over

# In python, you can assign multiple variables at the same time

## A,b = 100, 200

### A = 100 b = 200

# Apparently, you can also just append lists by adding them????

## a = [1,2,3,4]

## b = [5,6]

## print(a+b)

## [1, 2, 3, 4, 5, 6]

# |\*.\*||\*.\*|MUST RETURN BACK TO|\*.\*||\*.\*|CURRENT PROBLEM: <https://leetcode.com/problems/all-paths-from-source-to-target/discuss/471826/Python-dfs-solution>

## Given a directed, acyclic graph of N nodes. Find all possible paths from node 0 to node N-1, and return them in any order.

## <https://leetcode.com/problems/all-paths-from-source-to-target/discuss/118691/C%2B%2BPython-Backtracking>

### Explanation of DFS method and BIG O time space

## The graph is given as follows: the nodes are 0, 1, ..., graph.length - 1. graph[i] is a list of all nodes j for which the edge (i, j) exists.

## This solution was made by someone who reaaaaallly understood python well

### They used a deque – Double ENDED queue

#### Can add or remove from either end

### O(1) from either direction adding

### O(n) for pop and insert

## <https://www.geeksforgeeks.org/deque-in-python/>

# Same problem as above, but like easier: <https://leetcode.com/problems/all-paths-from-source-to-target/discuss/500839/Two-python-sol.-by-DFS-approach.-85%2B-With-Comment>

# |\*.\*|DFS|\*.\*| with a recursion, and then with a stack

# Get some dfs practice in yo, this stuff is still confusing for you at this time

# Recursion

## Get some DFS

# Stack

# Setting up at home

## Dark theme leetcode

### <https://github.com/CyC2018/Dark-Theme-Leetcode/blob/master/README.md>

## GitIgnore

### <https://intellij-support.jetbrains.com/hc/en-us/community/posts/115000154070-How-to-gitignore-idea-files>

### <http://egorsmirnov.me/2015/05/04/global-gitignore-file.html>

## Go through and fix Search Everywhere, and Evaluate Expression

## Add ## binding

# Kadane’s algorithm

## Finding the maximum subarray value in an array

## <https://www.youtube.com/watch?v=86CQq3pKSUw>

## Look at each index

### Ask ourselves what is the max subarray at this index?

### What is the max at src[:1], src[:2], src[:3] etc.

## Brute force alone will be O(n\*\*2)

## THE TRICK is where you realize the max of something like src[:3] is impacted by src[:2]

### Since src[:3] = src[:2] max + src[3]