Small Code Challenge Problems

- 20 - 45 minutes

- typical solutions: 10-40 LOC

- used extensively in interviews for a reason

- mastery on a language

- logic / reasoning

- communications

- not a skill that you "acquire and file away", but needs a lot of practice

Understand the Problem

- requirements are explicit

- take notes

- the odd words problem

- requirements are not so explicit and need to be modeled

- examples need to be described in computational words

- diamond of stars

- implicit knowledge needs to be captured

- convert to explicit rules, in computational language

- what century is that

- identifying and defining important terms and concepts

- queen attack

- same row; same column; esp. same diagnal

- ask questions to verify your understanding!

Examples / Test Cases

- Input / Output

- Test cases serve two purposes:

- help you understand the problem

- allow you to verify your solution

- happy paths

- combination of requirements; the "obvious" result

- Edge cases

- focus on input

- emptiness (nil/null, "", [], {})

- boundary conditions

- repetition / duplication

- data types

- Failures / Bad Input

- raise exceptions / report errors

- return a special value (nil/null, 0, "", [] etc)

- ask questions to verify your understanding!

Data Structure

- input data

- rules/requirements as data

- hash / object

- string, array, hash/object, number

- string

- concact, strip, reverse, etc

- Regular Expression! split, replace, match...

- array

- need to walk through it (iteration)

- index

- abstractions!!

- map

- reduce

- select/filter

- all

- ...

- hash/object

- lookup table / dictionary

- partition data for more efficient access downstream

- digest

- number

- math operations

- number as string may have advantage over numbers

- compound data structures

- array of arrays

- hash with array/object values, etc

Algorithm

- Algorithms have to be described in the language of chosen data structure!

- "then solve it" doesn't count

- Have to be really fluent with

- String / Regex

- Array

- Ruby: Enumerable

- JavaScript: Higher Order Functions

- Hash / Object

- Creation (default values)

- Access (default values)

- Iteration

- verify your algorithm with your examples / test cases

Abstraction

- Very important, rely on language built in abstractions: String/Regex, Array, Hash/Object

- Avoid solving big problems!

- Always, always, break bigger problems into smaller problems

- Don't try to walk through a wall

- lay out the general steps of an algorithm, without having to go to details

- Try to solve a problem in 1-2 sentences!!!

- If you can't, raise your abstraction

- create helper methods/functions

- push detailed steps to methods/functions

Interview Tips

- Communicative

- Getting to a working solution >> no solution

- De-risking

- Check your building blocks

- run code often

- debug locally

- Manage your energy