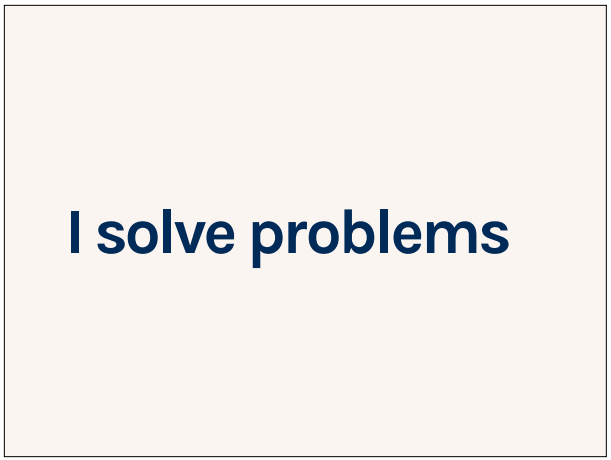
1

✨ Hello to readers from You Got This! Thank you for being part of such a wonderful event.

I have one quote from mathematician George Polya in this deck, but I can't emphasize enough how interesting and helpful I've found his *How To Solve It*. While a lot of this talk draws on my own experience, *How To Solve It* has provided tons of insight and food for thought. See: https://press.princeton.edu/titles/669.html You **do** want the edition with the Conway introduction.

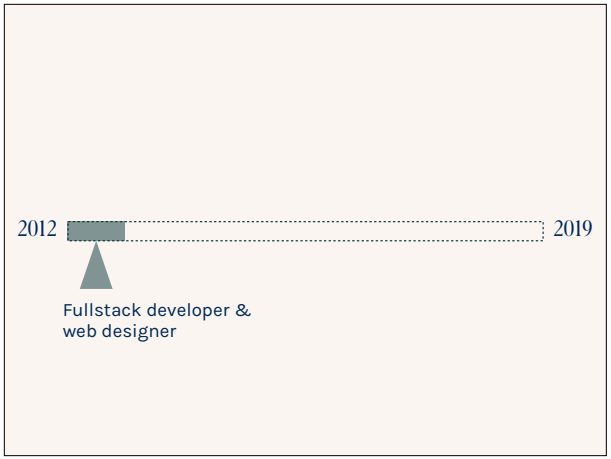
P.S. — my writing and punctuation style here is wonky because I was trying to give myself speech cues. I do actually know how commas work ;)

———————



2

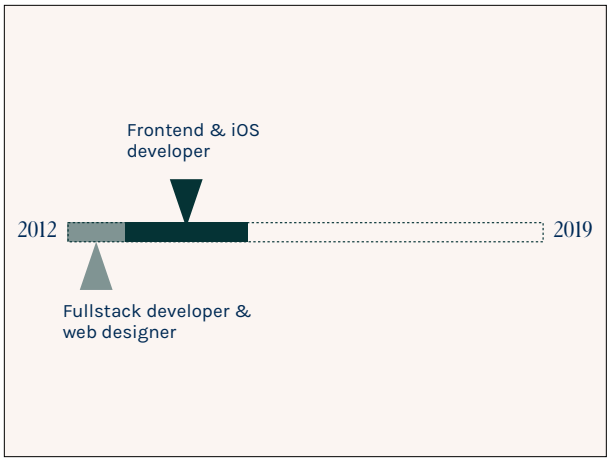
So, to begin with, my name is Violet, and I solve problems. Mostly, web development problems.



3

I've worked in this field since 2012, when I graduated from college with a degree in Spanish literature.

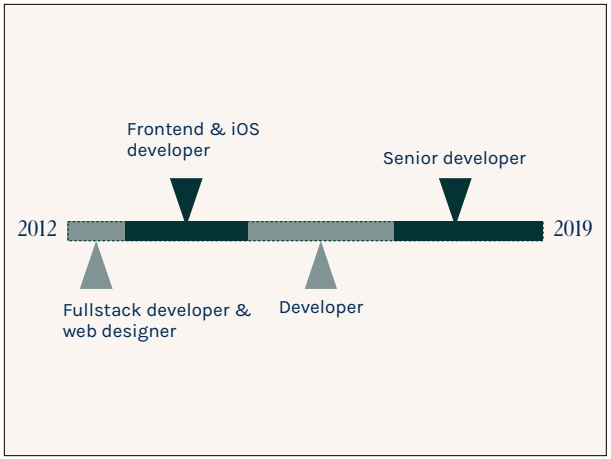
After I graduation, I stayed in town for 7 months in my first full-time development job. I was part of a three-person dev team, and all of us did backend, frontend, and design.



4

After that, I was a frontend developer at an energy-efficiency startup in New York City. There, I was part of a six-person engineering team, but I was the only frontend developer; everyone worked on software or databases.

My job ranged from making tiny static sites to developing new features on our actual product site to building a Shopify store for the company. I also somehow ended up being our iOS developer, which at the time was completely new to me.

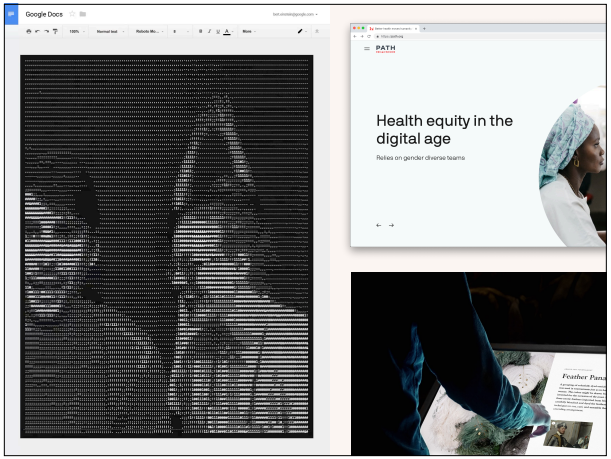


5

After almost two years there, I moved across the country to Portland, OR for a job at Instrument, a creative agency. And there, the *amount* of things I've worked on — and the amount I've learned — has taken off. Like, I can't even begin to fit it on that slide.

6

To start with, over the past four years, I've made things for tech startups, museums, sportswear companies, and nonprofits.



7

I’ve helped create style guides and component libraries, interactive installations and museum labels, websites with five pages or tens of thousands.

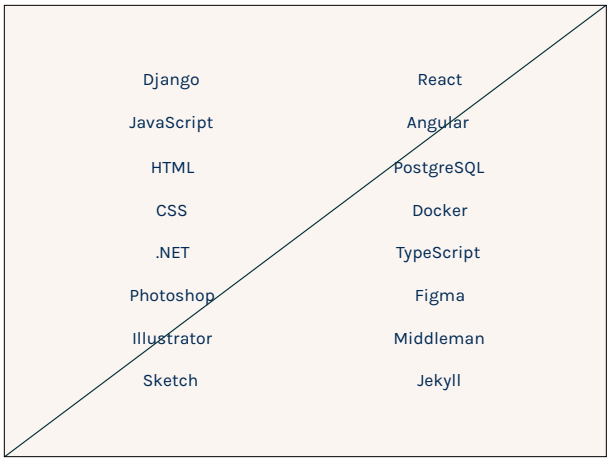
Sometime I’ve worked alone; other times, I’ve been one of seven developers.

And, on top of all of that, my role on the team has been dynamic. Sometimes I’ve been an individual contributor, focused on building out the project itself. Other times, I’ve been a tech lead — managing goals, making technical decisions, and supporting the team.

8

So, clearly, I love variety.

But the one constant amidst all of this is that my work is always getting more interesting. I’ve always got new challenges to take on, new ways to create and be creative. And that’s right where I want to be.



9

But the fact is, no single technical thing is **the** reason why I enjoy being a developer, and no **one** thing has carried me to this point in my career.

Independent Problem-Solving

10

Except, perhaps, for independent problem-solving.

This means: defining challenges, inventing solutions, finding connections, and getting a little better at it each time.

Independent problem-solving underlies pretty much all the work I've ever done, and most likely, yours as well.

We Are Creative Problem-Solvers

11

Because we are creative problem-solvers. Above all else.

We *do* specialize. We become experts in interaction, performance, typography. We lose ourselves in color theory and stack traces. But we're more than that.

Our true strength lies in thinking creatively. In finding answers. Figuring things out. Evolving.

And as such, problem-solving is a core competency for anyone in this room.

Reframe Independence

A Problem-Solving Framework

12

My goal today is to lay out for you some things that it took me years to piece together for myself.

We're going to do two things: Redefine independence, and look at a framework for independent problem-solving.

Reframe Independence

13

So, let's start by looking at what independence actually *is*.

Constructing understanding & solutions

14

For me, it's most useful to think of independence as this: the ability to construct understanding and solutions.

I say "construct" because these acts take *agentive work*. That work may happen in the moment, or it may be prefaced by decades of experience. But understanding problems (and oneself) doesn't happen without effort.

Likewise, we don't usually "find" a solution. Maybe we discover pieces of it, but solutions, like problems, must be purposefully brought into being.

~~working alone~~
~~getting it right~~

15

Notice that this definition says **nothing** about working alone or not needing help. Seeking help is actually a central part of independent problem-solving, as we'll see later.

Similarly, independence doesn't mean that you always know the answer or never need do-overs. What matters is that through it all, you are able to expand, understand, and improve.

Reframe Independence

A Problem-Solving Framework

16

So let's move on to the actual *act* of independent problem-solving. How can we think about this, and how can we be most effective when we're doing it?

A Problem-Solving Framework

17

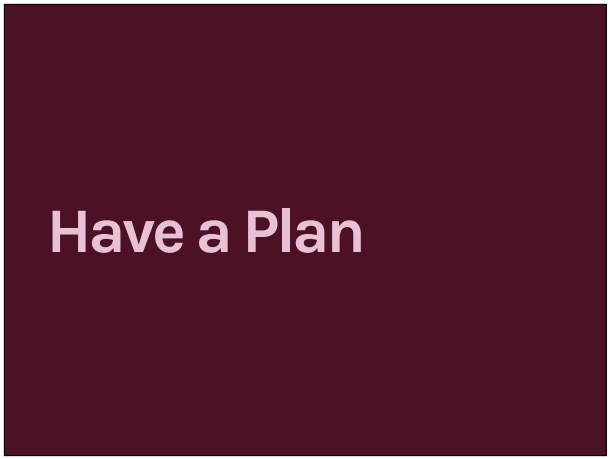
One thing to note about this next section is that these are all mental and behavioral strategies. As we go through them, some strategies may feel familiar, or resonate with you. They may already be part of your process.

For any **unfamiliar** strategies, the aim is for you to understand their value and incorporate them into your practice. Over time, they will become second nature.

Have a Plan
Be Honest
Use Your Tools

18

These problem-solving strategies fit into three broad recommendations: have a plan, be honest, and use your tools. Within each of these categories, we're going to look at specific ways to level up our independence and effectiveness.



19

So, let’s start with planning. Why is this important and what strategies here can help us?

20

When something new and exciting comes along, it can be tempting to dive right in and start… just, trying things. This might work occasionally, but it doesn’t really help us deliver consistent results, or learn over time.

It’s better to, before starting work, make a plan of how to tackle the problem.

21

It is foolish to answer a question that you do not understand.

— *George Polya, How to Solve It*

And before we can even **make** a plan, we need to **understand** the problem, as well as our end goal.

As mathematician George Polya puts it, it is foolish to answer a question that you do not understand.

22

Understanding

So let's begin by building our understanding. It's time to clarify what the problem is, check assumptions, and ask questions.

Here are some ways to check for and expand your understanding of the problem.

23

Understanding

- **Data** - What information or resources do I have? Is anything else helpful or necessary?
- **Unknown** - What is the task that must be accomplished? What is the problem that needs to be solved?
- **Condition** - What else must our solution do? What requirements should it satisfy?

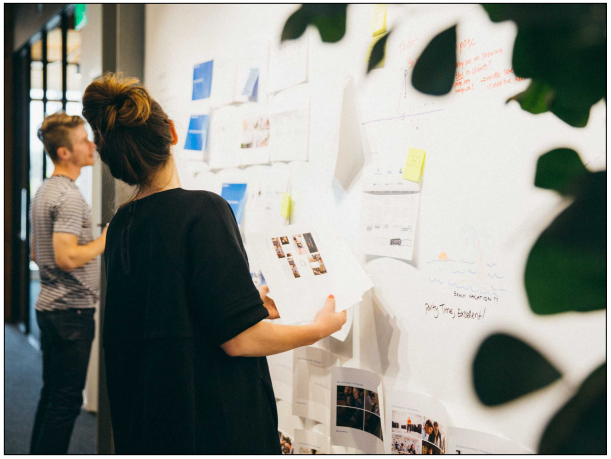
Start by looking at the **data**. What do you know about the problem, and is there any more information that you need, or would like to have?

Next, check the **unknown**. Clearly articulate what problem needs solving.

Finally, the **condition**. What else does our solution need to do? What other boxes should it check?

This a great place to involve your team members, who may have different context or a different understanding of the problem. Asking these questions together ensures that everyone is in agreement about the work being done.

24



At Instrument, understanding is so important that most projects begin with three to eight weeks during which we don't actually work on the deliverable at all. Instead of immediately working working working, it's our job during this "discovery phase" to interview stakeholders, clarify client needs, and reach agreement on the project's larger goals.

And this seriously *does* take eight weeks sometimes.

But, it ensures that when we *do* start work, we make something that everyone actually wants.

Planning

25

Now that we understand the goal, let's plan out how to solve or create it.

Maybe we already track tasks and to-dos in a ticketing system, or in post-its.

Planning

- How long will this take?
- What are **signs of progress?**

26

But we can vastly enhance the humble to-do list by estimating how *long* each step will take, as well as that step's *signs of progress.*

27

Signs of Progress

- How do I know that this step worked?
- How does this step push the solution forward?

A sign of progress is a way to know that you've advanced towards your goal. For a step on your checklist, how do you know that it worked, or that you're closer to completion?

28

```
describe(Class Dog):
× expect(Dog.bark()).toEqual('woof!') FAIL
× expect(Dog.getIsGoodBoy()).toBeTrue() FAIL
```

For example, the programming methodology of test-driven development is centered around signs of progress. The way test-driven development works is that, before writing any actual code, you write tests describing the behavior you want your code to ultimately have.

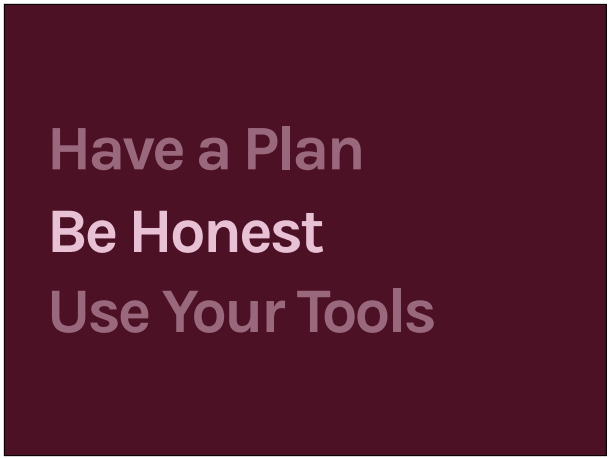
Here's some pseudocode for the Dog class we're making. We want a Dog to go "woof" when it barks, and we want it to be a good boy. For now, both of these tests will fail, since we haven't started coding the Dog yet.

29

But, once we add the right "bark" functionality, the first test passes. The passing test is a very clear **sign of progress** that we're on the right track.

30

And once we've written the code so that every test passes, we know we've reached out goal.



31

Great, so now we've got our plan. We understand the problem and have established signs of progress.

But I mentioned this thing about honesty. Where does that come into play?

32

Honesty is extremely important while you are actually solving a problem. Having a plan, even a great one, doesn't mean that carrying out that plan will be quick or easy.

As you work, check in with yourself, and be realistic:

is this step taking as long as you thought that it would? are you seeing the right signs of progress? and, lastly, how are you *feeling* about that step or about your solution in general?

33

Be Honest With Yourself

- Is this taking as long to accomplish as I said it would?
- Am I seeing the expected signs of progress?
- How am I **feeling**?

By asking yourself these questions and checking in frequently, you can avoid spinning on one minute aspect of the task, or straying off into a similar — but unrelated — problem.

You also build mindfulness around your own work habits, and these checkins will become easier as you examine those habits more closely.

34

Be Honest With Your Team

- “I am having trouble solving this.”
- “I am going to need more time.”
- “I will try a different approach.”

In executing a solution, this is also a time to be very honest with your team. And I am going to say the quotes below, one at a time, and I’d like you to repeat after me. These can be really hard to say in the moment, and I want to give you some practice.

So, let’s go:

I am having trouble solving this.
I am going to need more time.
I will try a different approach.

35

Be Honest With Your Team

- “I am having trouble solving this.”
- “I am going to need more time.”
- “I will try a different approach.”

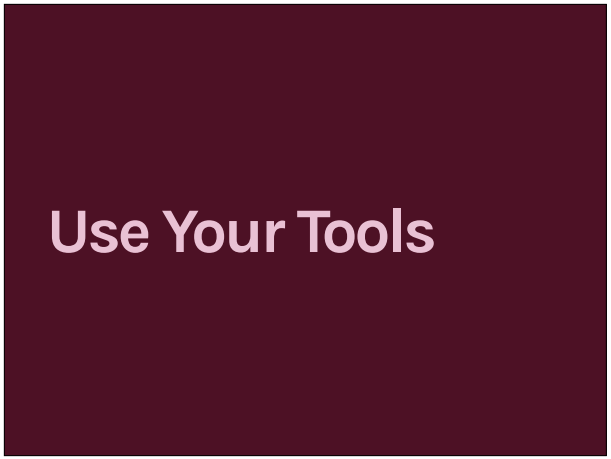
Thank you, thank you, you are lovely people. And right now, you’re really far ahead of me. It took me… four-ish, five ish years to become even vaguely comfortable with *expressing that I was having trouble*.

But the secret here is, managers **love it** when you go to them and are able to tell them what’s wrong and what you need to overcome it. It’s proactive, it’s good information for them, and it shows that you care about the project and not about looking perfect.

36

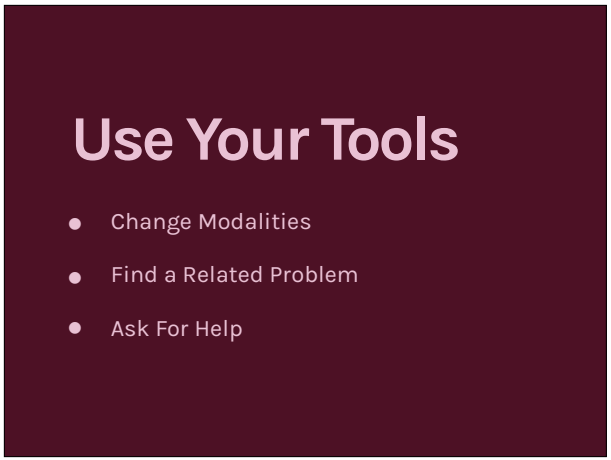
Have a Plan
Be Honest
Use Your Tools

And now, we’ve come to the last aspect of problem-solving: Use your tools.



37

By “Use your tools”, I mean, look to connect what’s happening now with things you already know or have seen. Mobilizing your knowledge, finding these parallels, can guide you towards steps and solutions.



38

There are three extremely helpful ways to leverage outside knowledge and experience: by changing modalities, finding a related problem, and asking for help.

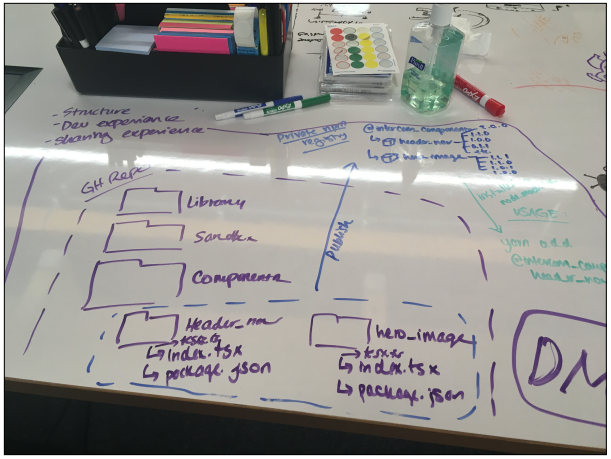
39

Change Modalities

- write it down!
- draw a picture!
- talk it out!

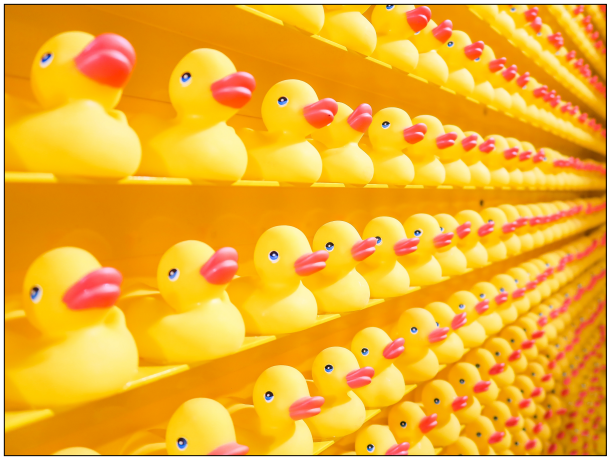
Changing modalities means switching up the **medium** you're using. We spend a lot of time in the mouse-keyboard-screen triangle, and while these are great tools, they can **trap** us in the same perceptions and thought patterns. If we're having trouble thinking something through, or finding an alternate solution, it might be time to express those ideas some other way.

40



Writing something on paper instead of typing it can be effective. I **love** drawing pictures and I often work through problems visually on a whiteboard before taking them back to my computer to type up.

Switching modalities helps awaken dormant associations and analogies. Something that's hard to express in words, for instance, may be easy to show spatially. Something that's easy to miss on a computer screen may completely stand out if it's printed.



41

One classic modality change is rubber ducking.

In programming, “rubber ducking” is a debugging tactic where you narrate every single thing that your code is doing, or everything that you’re trying out. It works regardless of if you’re talking to another skilled programmer or a literal rubber duckie sitting on your desk, which is where the term comes from.

Rubber ducking’s effectiveness comes not from actually getting feedback, but just from being forced to vocalize your ideas linearly. Assumptions, mistakes, and subtleties which may be easy to miss in written from are suddenly, **super** apparent.

Find a Related Problem

42

Another way to bring fresh knowledge into your current task is by finding a related problem.

everything is connected, maaaaaaannn

43

All problems are related to *some* other problem, by the way.

Find a Related Problem

44

Related problems are wonderful, because you've seen them before, and may have actually solved them, as well.

Related problems can:
- deepen our understanding of current problem
- suggest steps to take, or signs of progress
- they can point to alternate solutions, and
- they can illuminate domain-specific knowledge and "gotchas"

45

Find a Related Problem

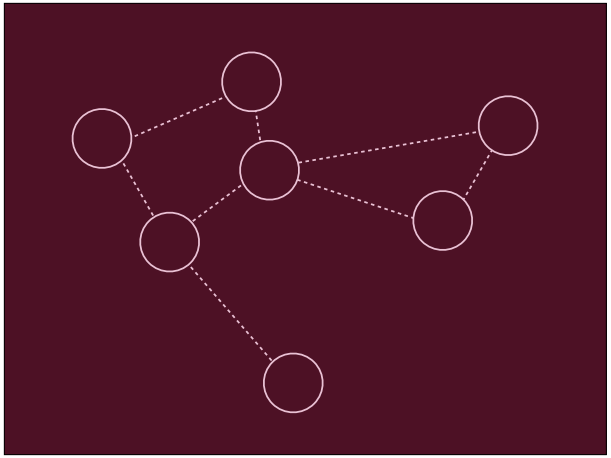
- **Data** - have I used these tools before? have I done something else with the same information that I have now?
- **Unknown** - have I made a similar end product before?
- **Condition** - have I made something that needed to satisfy similar requirements?

Problems can be related to each other in a variety of ways. If you remember back to when we were trying to understand a **single** problem, we can ask similar questions to find **related** problems.

Problems can be similar because of their data, unknown, or their condition.

Data-wise, a sidebar built in React may have information useful to an image gallery built in React.
Looking at an unknown, learnings from making a footer may be applicable to making a header, since they both solve navigation needs.
And having made an e-commerce site can inform how you build a blog, if they both need to satisfy the condition of working without JavaScript.

46



If this all sounds like a bit much, the good news is that as you gain experience, you'll have more and more solutions to look back on and use as related problems.

Taking on a wide variety of challenges can help you build up this stock of related problems to refer to later. And as you work through these problems, you'll start being able to notice and leverage patterns.

Asking for Help

47

You also have the option of bringing others' experience to the table. Asking for help can work wonders, but getting comfortable with it can be difficult.

So we're going to look at some tactics for seeking help. These tactics aim to structure how you seek help, build your confidence, and to make it easier for others to help you.

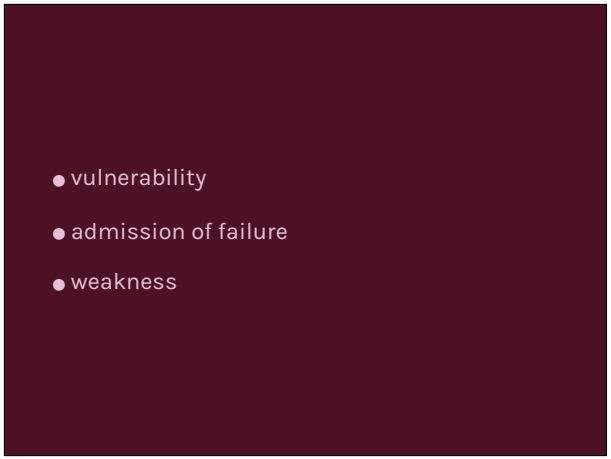
But First, a Secret

48

Before we dive into the specifics, though, I want to tell you a secret. A secret that… some of the talks today have also highlighted, actually. but it can't be said enough.

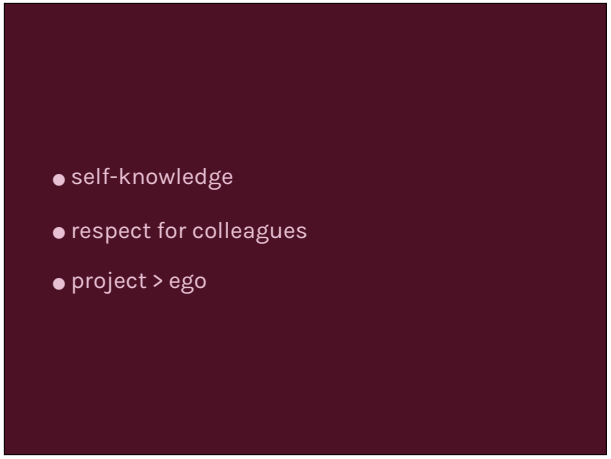
If you get comfortable asking for help, you will officially be ahead of **many** of your more senior colleagues.

For a variety of reasons, societal and personal, people don't ask for help, and… they get **used** to life that way.



49

If independence means that you don't need help, it follows that seeking help means that you are *not* independent. Needing help — accepting that you can't do something alone — can feel like admitting that you've failed. I know, because I've suffocated under that weight.



50

So here's what I wish I'd known earlier about asking for help.

Seeking help shows self-knowledge in that you are aware of your own limits — while you **should** be challenging yourself, you're not magic, and you won't gain omniscience through the same repeated struggle.

It shows that you respect colleagues and their expertise.

Most importantly, it shows that you value solving the problem at hand over preserving your own ego. This factor is *incredibly* important to your manager especially, and shows that you can be depended on.

51

Asking for Help

- Re-use solution that already exists
- Unlock other perspectives & knowledge
- Reveal arcane or hard-to-Google knowledge

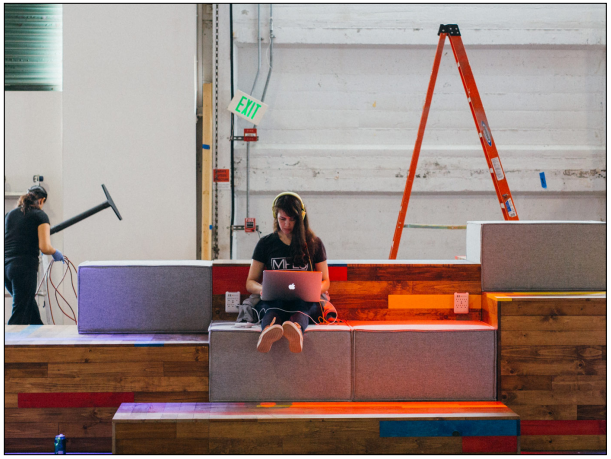
So, let's establish some really valuable things that you can get out of seeking help.

First, if someone has solved a **very** similar problem, you may straight-up be able to reuse their solution, simply tailoring the specifics, to your situation.

You can also use others' experience as a way to find related problems, and challenge your **own** assumptions.

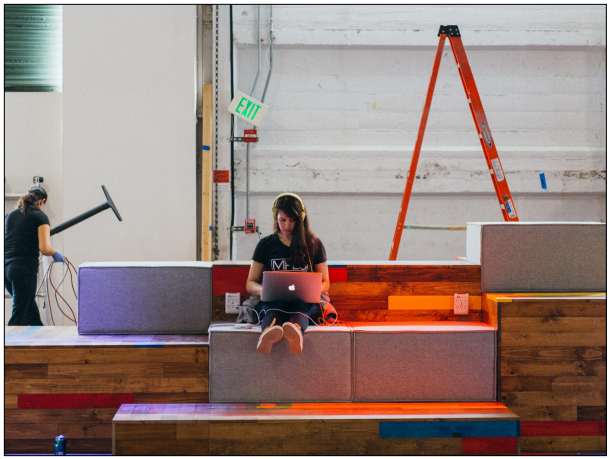
Lastly, some things are just really hard to isolate or Google. Asking another person can be the best way to move forward in a reasonable time frame.

52



So, for a personal example, here is a picture of me *not* asking for help. I don't think you can tell from the photo, but I am completely and totally **freaking out** on the inside.

Long story short, we were setting up an installation the day before an event, and for some reason, an animation that had worked fine through months of development and testing was suddenly dropping frames, all over the place. It was really obvious, and it looked. terrible.



53

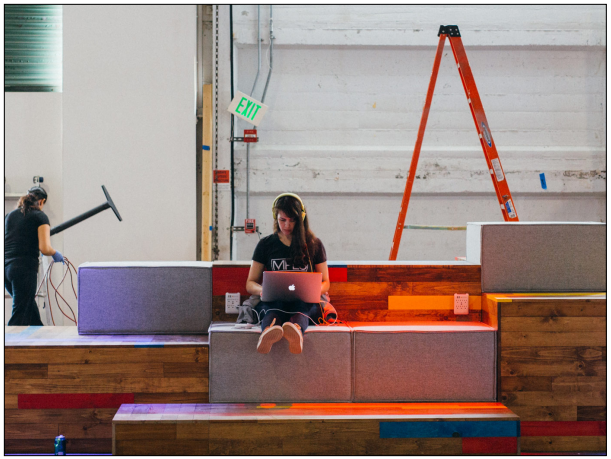
It was **my** animation, so, went my logic, it was on me to fix it, but I had no idea why it was happening. I holed up apart from the other, like, six developers who were there that day, and I just… Googled things over and over.

I Googled everything I could think of, and I was completely panicking that this was it, I wasn't gonna get it working again, I was gonna get fired. And I was already wondering if my parents would let me move back in with them while I looked for a new job… when a colleague wandered over, glanced at the problem, and suggested one line of CSS that I had never even heard of.

```
backface-
visibility: none;
```

54

In case you're wondering, that line was "backface-visibility: none".



55

Anyway, the moral is that if I had asked for help earlier, maybe this photo would have been of me eating nachos, or hanging out with my coworkers. Instead, I'm tense, worried, and alone.

So I want to help you maximize your nachos and hangouts, and minimize your worry.

Asking for Help: Who?

- Teammates, managers, coworkers
- Special-interest groups; e.g., on Slack
- **The Dream:** build a Voltron (thanks, Lara Hogan!)

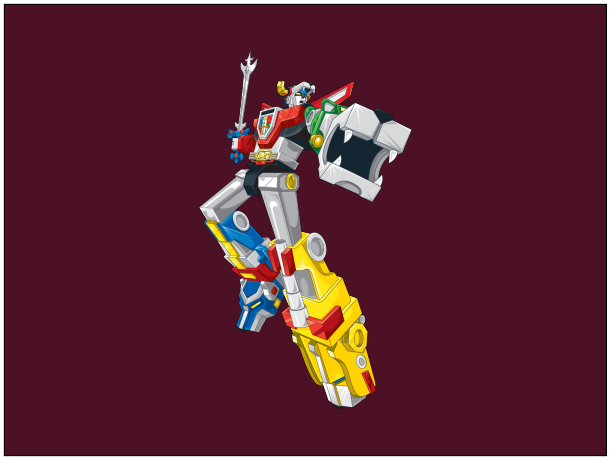
56

So, to begin with, who are good people to ask for help?

The answer is: pretty much anyone. Teammates and coworkers may have insight into your specific situation, or into the company standards and prior work.

Special-interest communities can help you get deep expertise on narrow topics. Slack groups are great for this. For example, I'm a member of an accessibility Slack group, and the other people in it are either experts, or at least care a whole bunch. The discussion and knowledge-sharing there is invaluable. So I encourage you to find and engage with, as you can, specific groups that correspond to your interests.

In the end, to borrow a metaphor from Lara Hogan, you want to build a Voltron.



57

Meaning, you should strive to build up a mental database of people you know who can help you with various topics. For example, right now my Voltron includes people who can help me with accessibility, technical design documents, TypeScript, and more.

⚡ See: Lara Hogan's Voltron post https://larahogan.me/blog/manager-voltron/.

Also OMG just read everything she's ever written. She's wonderful.

58

Asking for Help: When?

- if you cannot complete a step in your plan
- if you are not seeing a sign of progress
- if you need to change a plan or approach

So, now that you have someone to go to, when do you actually seek help?

Some good moments are: when you are unable to complete a step in your plan, or if you are not seeing an expected sign of progress. If, as you are carrying out a solution, you realize that you need to revise your plan, this is also a good time to seek advice.

The More Certain You Are That Someone Else Has Encountered This Before, the Earlier You Can Seek Help

59

Here's one of my rules of thumb for seeking help:

if I'm pretty sure that someone else has dealt with the exact thing that I'm trying to solve, I ask for help very early on. They probably won't need to take much time or think very hard, and will likely copy-paste some code and send it to me.

On the other hand, if I'm pretty sure that *no one* in my usual group has done something similar, I'll work on it for longer before seeking help. In this case, giving me help will be a larger investment of time and effort for the other person, and I want to make it worth their while.

Asking for Help: How?

- Give context
- Ask a specific question
- Describe your progress
- Say thanks!

60

Which leads me to the last part of this strategy — *how* to ask for help. These are guidelines, not hard-and-fast rules, but they do make it easier for others to help you.

Give Context

61

First, it helps to give context. Some questions are okay to ask with no preamble, but most of the time, it helps to know why you're asking the question in the first place. This helps colleagues understand your position and make better recommendations.

Ask a Specific Question

62

Sometimes, and this happens to everyone, you're lost enough that you just need some general guidance on a topic. Most of the time, however, it's to your advantage to ask a specific question.

What Have You Already Tried?

63

Listing out what you've already tried or are trying cuts down on back-and-forth and helps others target their advice better. This way, they won't suggest something that you've already attempted.

Say Thanks

64

And lastly, it's just nice to end with a thank-you. This will help keep the virtuous cycle going by showing others that their time is appreciated.

Have a Plan
Be Honest
Use Your Tools

65

And with those pointers on asking for help, we wrap up problem-solving.

Have a plan, be honest, and use your tools. Those are the best skills that any of us in this field can develop.

66

I hope you are able to bring them into your own work, your own practice.

As you gain experience solving problems, you'll be able to push harder on your own boundaries, open up new creative possibilities.

Thank You :)

Say hi!
https://violet.is
violetpena@gmail.com

Work with me!
https://www.instrument.com/careers/digitaldesigner
https://www.instrument.com/careers/full-stack-developer

Thanks to
Thomas, Ginger, J5, Daniel

So thanks so much again, You Got This.

I’m excited to see how you grow, and how you create.

So, come say hi, shoot me an email, come work with me.

And in the meantime, happy problem-solving.