Modern Developer Workflow - 21

21) Mobile-first Essential
- "Mobile-first" means diff things in different context
- Begin writing CSS the "mobile-first" way.

Q: what are RWD? Shortcomings of traditional RWD? (Laptop → Tablet → Mobile).

- 2 Factors to "mobile-first" than "Desktop-first"
  - More traffic from mobile devices, your primary source of traffic should not be an afterthought.
  - Traditional "desktop-first" responsive design resulted in bloated, slow loading websites. (forcing smartphones to download huge image files that were intended for desktop monitors)
    - We don't want anyone to have to download extra data that they won't even use.
    - Mobile-first = efficient websites that load as quickly as possible for everyone. Begin with the most essentials, then progressively enhance the layout and the assets only when necessary for larger, more advanced devices.

Q: "Mobile-first" in diff. context

Design	Development
- Design..	- Code the site so devices don't download unnecessary data
- Force to prioritize content	- Treat the most essential view as the baseline & code upwards
- Anticipate most common user actions	- Make the site load quickly for everyone
- eg. Info architect, graphic designers, usability experts.	

* Not all projects you'll work on will include a mobile-view design comp, but NOT AN EXCUSE! Even if we are only given a desktop design comp, there is no reason to not make every website you develop a mobile-first website. We are not going to let anything get in the way of building a website that loads quickly on all devices.

where to begin with mobile-first ...ev?
└ CSS

1) npm run dev → localhost:3000.
2) _large-hero.css
   @media (min-width: 530px) { }

→ A PostCSS tip that will make working with media queries much easier
= Mixins
1) Configure postCSS to support mixins
2) npm install postcss-mixins --save-dev
3) In webpack.config.js:
   └ require('postcss-mixins'),
4) npm run dev

3) app > assets > styles > base > _mixins.css
   └ @define-mixin atSmall { } ← name of mixins
      └ @media (min-width: 530px) { }
         └ @mixin-content;

4) In style.css
   └ @import "base/_mixins";

5) del. @
   &__title { }
      └ @mixin atSmall {
         font-size: 4.8rem;
      }
   &__subtitle {
      @mixin atSmall {
         font-size: 2.9rem;
      }
   }

6) In _mixins.css:
   @define-mixin atMedium {
      @media (min-width: 800px) {
         @mixin-content;
      }
   }

   @define-mixin atLarge {
      @media (min-width: 1200px) {
         @mixin-content;
      }
   }

6) In _large-hero.css,

	at Small	at Medium	at Large
h_title (font-size)	1.1rem	> 2rem	> 3.2rem > 4.8rem

22) Responsive images

1) What & Why? → diff. image for diff. screen sizes

2) Eg. 2 unique situations (2 unique code patterns).

└ ≈ traditional <img> (2 problem : data ; image-crop)

└) Situations to use a responsive images & diff codes.

└ a) Art direction & cropping situation

b) Image resolution & file size situation (faster load times)

└ → m-crop-small ; m-crop-medium ; m-crop-large

a) 1) del <img>

2) To send diff. images to diff users :: art direction / cropping reasons, use

```
<picture>
  <img src="images/m-small.jpg" alt="Puppy">
  <source media="(min-width: 760px)"
          srcset="images/dog m-medium"
  <source srcset="m-large.jpg" media="(min-width: 760px)">
```

(so this is how you create responsive img where you want to have fine grained control over how your images are being cropped at diff. screen sizes.

b) <img srcset="images/m-small.jpg, images/m-medium.jpg, images/m-large.jpg",

same crop, diff size

& don't have to write media queries; we don't have any opinion whatsoever about when the browser or device should use each one of these images. The browser & the device can figure that out on their own right because devices are self-aware. They know their screen size & the pixel density of their screen. They are intelligent enough to choose smallest image file that will still result in a crisp picture

↳ However, web browsers can figure out the size of an image but not until it's downloaded it → defeat the entire purpose of responsive image.

↳ Tell the browser the size of it

↳ <img src="~-small.jpg 570w, ~ 1200w, ~ 1920w", alt="Puppy ~">

↳ 570px wide

↳ As soon as the small img too small & it would begin to pixelate or blur, (GC) browser knows to switch to medium img to stay crisp.

↳ diff. devices & browsers have diff ways of determining when to use which image. This is a very hands-off approach. → Not trying to micromanage when the browser should use which image.

* if open in browser, starting with -large, won't narrow to -small. Because browser is intelligent enough to know if an img is sharp enuf for a large window, it's sharp enuf for a small window, → no sense to burn extra data to dl smaller images

↳ To check, < remove cache & visit from small size
< open in incognitive mode (large-img won't longer in the browser's cache)

1) npm run dev, localhost:3000

2) <picture>
  └ <img src="~--smaller.jpg" alt="~" >
  └ <source media="(min-width:640px)"
      srcset="~--small.jpg"
  └ <source srcset="~--medium.jpg" media="(min-width:990px)" >
  └ <~ ~--large ~ media="~ 1380px ~" >

Q: If using 4K/5K monitor, send an even larger image that is using the exact same cropping ratio.

↳ <source srcset="~--large" 1920w, ~--large-hi-dpi.jpg" 3840w >

} to medium, small, smaller
1380w 990w 640w
2760w 1980w 1280w

↳ send high dpi version if necessary.

23) Tips to test Responsive Images

↳ use -i copy
↳ "information" eg. 1920×654

Q: high dpi version?
↳ Inspect → Toggle device → ⋮ → Show device pixel ratio → DPR: 2.0 → Refresh

24) Creating Reusable Blocks

↳ Ask ourselves if the CSS property is unique to the block or it's a pattern that we could use again & again throughout the page

1) > modules > _wrapper.css → @import "modules/_wrapper";

```
.wrapper {
  padding: 0 18px;
}
```

2) In index.html:
↳

3) Replace mixins as

```
~~&~~ &__title{ → font-size: 2.4rem;
  @mixin atSmall {
    font-size: 4.8rem;
  }
}
```

```
&__description {
  font-size: 1.1rem
  @mixin atSmall {
    font-size: 1.875rem
  }
}
```

In _btn.css {

```
&--large{
  padding —
  @mixin atSmall {
    font-size: 1.25rem
  }
}
```

}

In _large-hero.css :

```
.large-hero{
  border-bottom: 10px solid $mainBlue
}
&__image {
  display: block; * → ** By default, web browser treat img element as inline level element, the browser thinks the img as a line of text, leaving some room at btm parts for letters
}
```

4) In html:
<img srcset="— smaller" — class="large-hero__image">

* "block" → treat the img like a div, not text.

Q: If vertical spacing between sections unique? or a design pattern that we could reuse throughout the webpage?

1) > modules > _page-section.css

↳ .page-section { } → @import "modules/_page-section";
  ↳ padding: 4.5rem 0;

↳

wrapper

In _wrapper.css {

+ → max-width: 1200px; 1112
  ↳ margin: 0 auto;
}

.page-section {
  &--blue {
    background-color: $mainBlue
    color: #FFF
  }
}

In html:

5) Headline Block

1) Create a generic, reusable headline block

> modules > _headline.css → @import "modules/_headline";
  ↳ .headline {
      font-weight: 300;
      font-size: 2.875rem
    } color: $mainBlue

2) modifier class: headline--centered.

.headline {
  &--centered {
    text-align: center

3) modifier class: headline--orange

  &--orange {
    color: $mainOrange
  }
}

4) modifier class: headline--small &--small {
  font-size: 1.875rem;
}

modifier class:

5) headline--narrow

```
&--narrow {
  max-width: 500px;
  margin: 0 auto;
}
```

6) modifier class: headline--light

```
&--light {
  font-weight: 100;
}
```

7) to bold.

8) modifier class: headline--b-margin-small

```
&--b-margin-small {
  margin-bottom: .5em;
}
```

headline--b-margin-large

```
&--b-margin-large {
  margin-bottom: 1.6em;
}
```

define margin on text elements in em instead of rem. em is relative to the font-size of the element, which means we could reuse this class on headlines that have diff. font-sizes and this will always scale nicely

Q: Too much modifiers? Why not few unique classes?

- Yes, absolutely.
- Why modifier system makes sense: on a larger website with hundreds of pages, the designer might regularly send us new styles to develop & if we are not using the modifier system, it can become exhausting to continually come up with a new unique class name. Creating single monolithic unique classes can be come exhausting & result in disorganized code. As those heading classes add up over time, eventually you'll reach a point where those class names don't mean anything to you. That are huge disorganized spaghetti CSS files that are impossible to maintain.

With modifier system, the designer could come years later with a new style to develop, we could just look at the new headline style & think

"Can we create this new style by combining existing modifiers?"

↳ if can → excellent

if can't → create a new modifier or two.

↳ we know exactly where to write, & our class names have inherent meaning.

eg: 1) <h2 class = "high-impact-headline" ... >

2) <h2 class = "headline headline--large headline--orange" >

↳ the modifier system will make sense to us even revisiting after years.

↳ these class names will always make sense & easy to reuse & recycle

∴ There's no right / wrong way to structure your CSS but give the whole BEM way of thinking about things. A real try before you dismiss it.

9) enclosing with

10) Modifier class: wrapper--medium

In _wrapper.css:

```
&--medium {
  max-width: 976px;
}
```

11) In _page-section.css:

```
.page-section {
  @mixin atMedium {
    padding: 4.5rem 0;
  }
+ → padding: 1.2rem 0;
```

decrease baseline font-size

In _headline.css:

```
.headline {
  @mixin atSmall {
    font-size: 2.875rem
  }
+ → font-size: 1.9rem

&--small {
+ → font-size: 1.2rem;
  @mixin atSmall {
    font-size: 1.875rem;
  }
}
```

12) .wrapper {

```
  .wrapper {
    padding: 0;
  }
}
```

→ any .wrapper that is nested within another wrapper should no longer have any horizontal padding.

---

26) Column Layout Block

width: 33.33% ...

---

29) Git Branches
└ what, create, switch, merge.

1) + to open new CL
2) git branch → master: the X commit work in progress code to the master branch
   new branch for each new feature
   git add . → to stage all files
   git reset ~.html → to unstage ~.html
3) * git push origin branch-1 → push to branch-1 branch on github. If not existed, create a new branch called branch-1 on github.
4) ~~To~~ git branch -d branch-1 → to delete branch-1

↳ other "branch topics:
↳ a) "--no-ff" merge option
   b) How to handle merge conflicts.

39) Object-Oriented Programming.

> App.js :

* `import '../styles/styles.css'` → add CSS
* `if (module.hot) { module.hot.accept() }` → enables hot module replacement

eg.
```
var kyle = {
    name : "Kyle Law",
    favoriteColor : "red",
    greet : function() {
        console.log("JunWei!!");
    }
}
kyle.greet(); → JunWei!!
```

OOP:
- Stop thinking in terms of individual, loose variables & function → Begin thinking in terms of cohesive, self-sufficient objects.

Object : an entity that has data & behavior (nouns & verbs)

* function inside of an object = method

↳ create a reusable blueprint / recipe that spells out what a person objects should be.

↳ by leveraging constructor function

eg → begin blueprint name with Capital Letter

```
function Person() {
    this.greet = function() {
        console.log("Hello there!");
    }
}
```

this. keyword, allows our blueprint function to be flexible, recyclable.

```
let john = new Person()
```

↳ an operator that will create a new instance of the Person object type.

↳ create a new object using Person blueprint

john.greet() → Hello there!

eg. function Person(fullName, favColor) {

```
    this.name = fullName;
    this.favoriteColor = favColor;
    this.greet = function(){
        console.log("Hello there! My name is " + this.name + " and my Fav is "
                    + this.favoriteColor);
    }
}
```

→ let john = new Person("John Doe", "blue");
let jane = new Person("Jane Smith", "green");

↳ called ~~Class~~ Class in ~~other~~ Java, C++, Python, but ~~Technically~~ technically, JavaScript doesn't have class.

40) The JS Module Pattern and "webpack"

↳ Split JS into multiple files.

1) scripts > modules > Person.js
   * move Person() constructor function
   ↳ export default Person

2) In App.js
   import Person from './modules/Person'

⇒ An alternate way to create a blueprint. ☺ Class syntax

1) class Person {
       constructor() { }
   → within a class, constructor() is a reserved, special word, JS will know to call the constructor when a new object is created using our class.
   ↳ create properties

2)
```
constructor(name, favoriteColor){
    this.name = name;
    this.favoriteColor = favoriteColor;
}
greet() {
    console.log(...)
}
}
```

* Behind the scenes, JS is still not using classes, it's not using class inheritance, it's still running based on prototypes.

- class make inheritance a piece of cake.

1) class Adult extends Person {
    payTaxes() {
      console.log(this.name + " owes u");
    }
  }

→ new class Adult which inherited everything from Person class,
  └ Avoid lots of duplication.

2) let jane = new Adult("Jane Smith", "green");
   jane.payTaxes(); → Jane Smith now owes ....

41) Applying our New JS knowledge.

1) del Person.js & everything related.
2) scripts > modules > MobileMenu.js
   └ class MobileMenu {}
   └ export default MobileMenu;

3) class MobileMenu {}
   └ constructor() {
       alert("Testing from MobileMenu.");
     }

4) In App.js ← just a variable name, could be anything.
   └ import MobileMenu from './modules/MobileMenu'
   └ new MobileMenu();
   let mobileMenu =

5) In browser, alert msg pops up !

6) ~~constructor{}~~
   ↳ should not do! (spaghetti code):
   └ constructor() {
       document.qs(".nav-btn") (1) .addEL("click", function() {}) (2) (3) -
       └ console.log("The top right icon was clicked");

   ↳ a plate of noodles, everything is sort of tangled & intertwined together

① Selecting Elements from the DOM
② Event handling
③ Defining functionality.

7) del.
→ constructor() {
  this.menuIcon = document.qs(".site-header__menu-icon")
  ↑ only include property/shortcut that stores DOM selection for the menu icon element inside {}
}
events() {
  this.menuIcon.addEL("click", () => this.toggleTheMenu())
  ↑ list any events that we want to watch for
  ↑ so that it does not manipulate the value of 'this' keyword.
  ↑ add EL is not receiving a direct reference to our method, instead we are passing out EL arrow function that's going to call
  ↑ arrow function is actually going to execute toggle the menu.
  ↑ anyname
}

toggleTheMenu() {
  console.log("w")
}

↑ As arrow function doesn't manipulate 'this' keyword, when toggleTheMenu() runs, 'this' keyword will still be pointing towards the overall object.

8) constructor() {
  +→ this.events()

9) when click on icon, add a new CSS class to the element that's currently hidden.
→ Add another property of the div. element that contains the content hidden for small screens.
constructor() {
  +→ this.menuContent = document.qs(".site-header__menu-content")
  ↳ the element we wanted to toggle a class to

10) toggleTheMenu() {
  this.menuContent.classList.toggle("site-header__menu-content--is-visible")
  ↳ without ⇒ ; this → element that got clicked on.
  ↳ with ⇒ ; this → overall object, which lets us easily access our menu content property

11) In _site-header.css :
  .--menu-content {
  +→ .--is-visible {
      display: block;
    }
  }

42) 1) .l--main-content {
+→ padding-top: 100px;
@mixin atMedium {
+→ padding-top: 0;

2) In _primary-nav.css:
li {
display: inline-block;
+→ @mixin atMedium {
+→ display: block; > padding-right: 20px;
}
}
li:last-child {
padding-right: 0;
@mixin atMedium {
padding-right: 20px;
}
}

3) .site-header {
+→ transition: background-color .3s ease-out; ##
.l--is-expanded {
background-color: rgba($mainBlue, .55);
}

In MobileMenu.js:
constructor() {
+→ this.siteHeader = document.qs(".site-header")
}
toggleTheMenu() {
this.siteHeader.classList.toggle("site-header--is-expanded")

4) `&__menu-content {` , `transform: scale(1.2);` } zoom out effect.

`@mixin atMedium {` `transition: all .3s ease-out;` }

```
transform: scale(1);
opacity: 1
```

`&__menu-icon {`

```
z-index: 10;
```

43. ☰ ⟶ X

1) In _site-header.css :

`&__menu-icon {`

del → ~~background-color: #FFF~~ x

```
width: 20px;
height: 19px; // subtle diff...
position: absolute;
z-index: 10;
top: 10px;
right: 10px;
```

`&__top {`

```
position: absolute
top: 0;
left: 0;
width: 20px;
height: 3px;
background-color: #FFF;
```

}

`&__middle {}` `top: 8px; // 8px down from top`

`&__bottom {}` → ~~top: 0~~ 2 `bottom: 0;`

`&::before {` → `content: "";`

`&::after {` → `content: "";`

Using 1 div :

In index.html :

```
<div class="site-header__menu-icon">
```

del → `<div class="site-header__menu-icon__top">` top

del → `<div>` middle

`<div>` bottom

2) In MobileMenu.js

```
toggleTheMenu() {
    this.menuIcon.classList.toggle("site-header__menu-icon--close-x")
```

In _site.header.css

```
&__menu-icon {
  &--close-x {
    &::before {
      transform: rotate(45deg) scaleX(1.25);
    }
    &::after {
      transform: rotate(-45deg) scaleX(1.25) translateY(1px);
    }
    .site-header__menu-icon__middle {
      opacity: 0;
      transform: scaleX(0);
    }
```

```
&::before {
    transform-origin: 0 0;
    transition: transform .3s ease-out;
}
```

horizontal vertical
50% 50% (default)
top left corner
longer

```
&::after {
    transform-origin: 0 100%;
    transition: transform .3s ease-out;
}
```

left bottom

```
&__middle {
    transition: all .3s ease-out;
    transform-origin: 0 50%;
}
```

---

43) Revealing an Element when scroll.

1) scripts > modules > RevealOnScroll.js

```
class RevealOnScroll {}
export default RevealOnScroll;
```

2) In App.js

```
import RevealOnScroll from './modules/RevealOnScroll'
let revealOnScroll = new RevealOnScroll();
```

3) In RevealOnScroll.js

```
class RevealOnScroll {
  constructor() {
    this.itemsToReveal = document.querySelectorAll(".feature-item")
  }
  hideInitially() {
    this.itemsToReveal.forEach(function(el) {
      el.classList.add("reveal-item")
    })
```

4) .forEach(el => el.classList.add("reveal-item"))

5) ~~In~~ constructor() {
+ → this.hideInitially()

6) styles > modules > _reveal-item.css @import "modules/_reveal-item";

```
.reveal-item {
  opacity: 0;   > transition: opacity 1.5s ease-out;
  &--is-visible {
    opacity: 1;
  }
}
```

7) Use JS to add --is-visible class to the hidden elements at precisely the right moment when they're scrolled to.

In RevealOnScroll { }

```
└ events() {
    window.addEL("scroll", () => {
      this.itemsToReveal.forEach(el => {
        this.calculateIfScrolledTo(el)
      })
```

> constructor() {
→ this.events()

~~calculateIfScrolledTo(el) {~~
~~console.log(el.getBoundingClient~~

8) calculateIfScrolledTo(el) {
- console.log(el.getBoundingClientRect().y)

→ perf in Browser

9) delete

let scrollPercent = () * 100
└ el.getBoundingClientRect().y / window.innerHeight
(webbrowser height)

if(scrollPercent < 75) {
el.classList.add("reveal-item--is-visible")

✓ but the codes are terribly, horrendously inefficient, waste the computing horsepower of the browser.

✓ Intersection Observer → allow us to code this feature while using much less computing resources.

---

51) Modal Section:

- modal.css:

```
.modal {
  position: fixed
  top: 0
  bottom: 0
  left: 0
  right: 0;
  background-color: rgba(255, 255, 255, 0.94);
  z-index: 5;
  display: flex;
  &__inner {
    flex: 1;
    margin: auto;
  }
```

```
&__close {
  position: absolute;
  top: 15px;
  right: 15px;
  font-size: 2rem;
  color: $mainBlue
  transform: scaleX(1.2);
  transform-origin: 100% 0;
  cursor: pointer

  &:hover {
    color: $mainOrange;
  }
```

(right, top)

6% see through

```
&__description {
  text-align: center;
  font-size: 1.3rem;
  font-weight: 300;
  line-height: 1.65;
```

```
.section-title {
  &--less-margin {
    margin-bottom: 1.3rem;
    margin-top: 0;
  }
}
```

```
.wrapper
  &--narrow {
    max-width: 975px;
  }
}
```

- social-icons.css

@import "modules/_social-icons";

```
text-align: center;

&__icon {
  background-color: $mainOrange;
  display: inline-block;
  width: 72px;
  height: 72px;
  margin: 0 5px 5px 5px;
  position: relative;
  img {
    display: block;
    position: absolute;
    top: 50%
    left: 50%
    transform: translate(-50%, -50%)
  }
```

65) Integrating React Into Workflow

↳ In real world, you're not always going to get to start from stratch, often you'll want to integrate react into a workflow that you've already set up.

React: An incredibly popular, in-demand, ~~FE too~~ lib can make our life easier when writing front-end JavaScript.

1) npm install react react-dom
   ↓ (core library) ↳ specifically for browsers environment
   ↳ no need '--save-dev' because they are direct dependencies that we want our visitors to actually download.

2) In App.js
   import React from 'react'
   import ReactDOM from 'react-dom'

↳ to actually output a React powered component onto our website, add an empty somewhere within HTML.

In index.html:
```
<div id="my-react-ex"></div>
```

3) In App.js
   ReactDOM.render(a, b)
   ↓ ↳ the element on the page that you want to render to. ≈ the empty div
   ↳ component you want to render to the page in ~~react~~. In react, everything boils down to the idea of components.

```
ReactDOM.render(<MyAmazingComponent />, document.qs("#my-react-ex"))
```

4) function MyAmazingComponent() {
   return (

   <h1> hi </h1>
    hi
)
   } Called JSX (≈HTML)

5) npm install @babel/preset-react --save-dev ↳ visitors don't have to download it, just for our workflow.

6) In JSX, use className. instead of class:

eg. <h1 className="..."> ...

7) Only 1 route top level element in JSX.

8) > MyAmazingReact.js

import React from 'react'

function ④ here

export default MyAmazingComponent.

⑤ In App.js

import MyAmazing... from './...④'

* Research: - props
- hooks / useState

---

66) Git - Handle (& Avoid) Merge Conflict

- Git is good at taking 2 versions of a project & merging them together..
  - ↳ git merge <branch-name>

Usually, Git can automatically merge our code. But, if we ask Git to read our minds ... → Merge Conflict

Prob1 (~~Fix~~) (outdated) : 1) git pull origin master (to update)
2) At EL, press [esc] → :wq → [enter]
3) git push origin master

Prob2 : 2: - merge conflict (commit different changes to the same line of code)
- ↳ computer can't decide for us. As a human being, we have to step in & make a judgment call

1) open index.html 2) fix 3) git add . / git commit -m "Resolved merge conflict" / git push origin master.

Tips to avoid:

1) Run 'git pull' command often
   - if you're working with a team of other developers, you want to constantly be pulling in their commits from the server, so that you're working with the latest code. Make
   - Make this a habit

2) Communicate with your team

3) Standardize white space settings in text editor [tab] key.
   - change it at btm right corner

4) Think of "git-merge" as a 2-way street. (wtf)
   - won't help you to avoid conflict outright.
   - Deal with any potential conflicts before it's time to actually Merge feature branch into master
   - "we can also merge master into feature branch"

---

62) Job Interviews ← Tips to land more interviews.
                   ← Tips to interview well.

- Tips to land more interviews:
  1) Consider Working Remotely
     - cast a wider net
     - You're developer, you work on a computer - you can work from anywhere.
     - Don't let location hold you back
     - Search for remote-focused job boards.
  2) Don't limit yourself to listed openings only.
     - You can create an opening
     - If you see a win/win situation - reach out.
     - Check on "Careers" pages on company websites

3) Call Their Bluff
- Try to sense when a job description is lying.
- Job descriptions can be unrealistic.
- As long as you have majority of the list, apply.
  - eg. "Full Stack Developer", lots use that phrase pretty lightly.
- Trust your gut - are you qualified
- Error on the side of overestimating yourself instead of underestimating yourself
- Don't let the job description intimidate you. Don't pre-reject yourself.
- "You miss 100% of the shots you don't take" - Wayne Gretzky

4) Customize your message for each job
- make minor adjustments
- Research the company
- What experiences, qualities, & traits are they looking for?
- Highlight these things in ur msg.
- How do you align with what they're looking for?
- Don't copy & paste a generic cover-letter message for all of your applications.

5) Keep ur message short & sweet
- But not so short that you don't look interested.
- Introduce yourself
- 1 paragraph about ur experiences & strengths
- 1 or 2 sentences about how you are a good fit.
  - Anything longer than that you run the risk of sounding self-important or like you're compensating for sth.

6) Create a personal website / portfolio.
- No excuse not to do this considering you can use GitHub pages
- Don't worry about the actual content
- The website itself is your portfolio, a chance for the company to see your code, to see that you understand best practices.
  - Is it responsive, mobile-first, load quickly, stay current?
- You'd be surprised how many applicants don't have a website.

7) Don't use templates on your personal site.
   - Templates aren't evil, just don't use it on your personal site.
   - Five star chefs don't use boxed cake or cake mixes.
   - Templating looks like you just wanted to finish the site in the most economical rushed way possible.

8) Demonstrate that you stay current
   - Tweet about new things that you're learning
   - Blog about what you're excited to learn next.
   - list new tech that you're interested in on ur personal site.

- Tips to Interview Well.

1) It's okay to be nervous
   - Relax, you're human
   - Everyone is nervous – you aren't judged for it.
   - You don't look as nervous as you feel inside.

2) Prime yourself before the interview.
   - You don't want your nerves to rob you of your personality.
   - Try to stay grounded in ur personality, own ur nervousness instead of it owning you.

3) Interview is a 2-way street.
   - An interview isn't you trying to convince them to hire you
   - The pressure isn't on you.
   - They are in the hot seat too.
   - Do you want to work for them?
   - Don't give away all of your power before an interview.
   - You're good enough.
   - Think of the interview as a compatibility test, a meeting where ~~[illegible]~~ if you can be friends. (positive vibe, kind, humble, speak confidently, just like what you'd do in a social situation)

4) Don't try to make yourself seem super intelligent
- Being likable is more important
  - eg. a genius, smartest applicant, but x laugh, x positive about anyone/anything. Even though he's the most skillful person interviewed by far, he just wasn't likeable. → we didn't pursue him.
- Be positive → leadership potential

5) Convey enthusiasm
- this industry ~~things~~ change super quickly, the best developers are the ones that love what they do because this leads to them spending time stay current.
- Is this "just a job" to you?
- We have the luxury of working in an area that we're interested in.
- Tons of people would love to be simulated by their work.
- Let gratitude & enthusiasm show mentally.

6) Don't be afraid to present yourself as a specialist
- Don't be afraid to admit that you don't know sth. It's a turnoff when an applicant tries to act like they know everything because I know they're lying.
- Be honest, presents yourself in a realistic light.
- Strengths & weaknesses, open up bit vulnerable, working on improvements

7) Building up ur tech vocab
- Familiarize urself with many diff tech as possible. (not learn, just be familiar with them)
- Using a tech word in a sentence correctly shows you're aware how things fit together

8) Prepare a custom question beforehand
- To show that you're picturing urself working there, genuinely interested & engaged.
- Don't ask the same generic question in all ur interviews.
- Research ur employers before interviews, & create a question.

9) The biggest challenge you'll face is running out of patience.
- Finding the right job takes time
- You're not "taking to long"
- You're good enough, you're just ran out of patience & are now questioning urself

Ex: Strengthening Your Patience
- It's normal for a job search to "take forever"
- Maybe you didn't want the jobs you've interviewed so far.
- Multiple interviews without a job offer can be rough.
- The right job will come along that makes you glad the other companies passed on you.

→ You'll get what you want
- You know ur strengths
- You would add value to any team

You can ...
- Contribute to a project via Git
- Write clean HTML
- Write super organized CSS
- Write modular, object-oriented JavaScript
- Leverage build tools like Webpack & Babel

↳ All you've to do is keep ur patience, stay positive

---

68) Career Progression — learn next career prog...
- Apply whatever you've learned.
- The developer spirit is to never stop learning, "level up"
  - JavaScript, JavaScript, & more JavaScript
    - More companies that will value JS, ∵ JS is now completely ubiquitous
    - FE, BE, simple sites, complex web apps, device-native apps
    - Always learn more about JS. Advancing ur knowledge, understanding on JS will propel ur career

- Learn what interests you the most. Your curiosity is the best motivator & teacher. Whatever interests you, pursue it from a JS angle.

  FE : React

  BE : Node.js

- Career Progression
  - Know Yourself
    - Everyone has their own unique blend of strengths & weaknesses
    - Align Your career path to leverage ur strengths
    - We are all good at diff things, we all have unique gifts
    - Aim ur career at ur gifts., aligned with ur natural gifts to remain energized.

- TIPS : ☆☆☆
  - 1) Limit ur complaining
    2) Try not to dodge responsibility.