UX/UI

For Researchers

Vincent Rubinetti
Software Engineering Team
Department of Biomedical Informatics
Anschutz School of Medicine
University of Colorado
Fall 2023

Intro

My work

Word Lapse

Preprint Similarity Search

Het.io and Connectivity Search

MyGeneset.info

Adage

Manubot

Monarch Initiative

Simplex

Human Microbiome Compendium

Lab Website Template

... and many other personal projects

I'm a UX/UI designer and developer with the Software Engineering Team for the department. I've worked with the Greene Lab extensively, as well as the TIS Lab, Krishnan lab, and external collaborators. You can take a look at what I've designed and built here to get a feel for my philosophies around this topic.

Disclaimer: A lot of this is just my view and experience. This topic involves a lot of things that are ultimately subjective, like organization of ideas. There are studies about these things, but not as much as in scientific fields, and it's all pretty dependent on culture and context. All of that said, I try to provide supporting references and some ways to feel more confident and objective about your decisions.

Overview

📜 Content

Tenets and advice

Common researcher pitfalls

Standard paradigms

Examples and exercises

~~HTML/CSS/JS~~

🎓 Takeaways

Better idea of how to think like a designer

Handful of tips to put into practice

Appreciation and respect for the craft

What will be contained in this presentation?

General principles that I try to keep in mind and emphasize to collaborators when I'm designing a project.

Pitfalls that I see researchers specifically fall into. There are certain habits from working in academia that become anti-patterns in the context of the web.

Some common patterns (tips and tricks basically) that you can follow to make your app look and feel good.

Some basic examples and an exercise at the end.

We will explicitly not cover the deep technical aspects of implementing a webapp, as it's outside the scope of this talk.

What do I hope you'll learn?

I want you to be better able to understand the mindset of a designer. What type of things factor into decisions, even if you can't remember the specifics.

I want to give you a handful of concrete things to walk away with that you can put into use.

Most importantly, I want you to care about UX/UI. It's not something you should put minimal effort into. "Good enough" is not good enough. People specialize in this field, and it has as much depth as any other field to an extent that you're most likely not aware of. Don't underestimate it. I've been doing this for many many years and I still feel far from an expert.

UX vs. UI

User E**x**perience	**U**ser **I**nterface
👆 The *feel*	👁️ The *look*
Information organization	Visual consistency and meaning
User workflows/paths	Polish, "pixel-perfect"
General behavior	General aesthetics
Research-focused	Design-focused
High level (site, pages)	Low level (components)

Ref Ref

UX and UI are sort of just what they sound like. UX is the feel, UI is the look.

UX involves deciding on the overall organization of information, how users move through your app, and how the app behaves. I'd describe UX as being a little more objective than UI in the sense that you can do research to try to measure it. For UX, you're generally looking at the bigger picture, at the app and page level.

UI involves deciding on a consistent visual "language" for the app to convey functionality and meaning, and then of course making it look good and polished. This tends to be more subjective and less measurable, and thus requires more of an artistic "eye". For UI, you're typically working at a lower level within the app, like on components (e.g. buttons, sliders, etc.), typography, iconography, etc.

There's certainly overlap and gray areas between these definitions, and a lot of people use them interchangeably. There's not a precise dictionary definition that everyone agrees on, but this distinction is generally what people think of.

They also sort of affect each other. UI definitely affects how a user experiences the app, and a UX goal might inform how the UI looks. I tend to think of UX as a superset of UI since it's just a more broad and encompassing term.

Side note: you'll hear me use the terms "(web)site" and "(web)app". Website typically connotes more static content, whereas as webapp typically connotes more dynamic/interactive content, but the overlap between these days is large and the terms can sort of be used interchangeably.

UX Process

6

Let's talk about a general process you can follow to design a project.

These are *roughly* in the order, but it doesn't have to be rigid. You can certainly rearrange these steps and even repeat and iterate on them multiple times. But I do think the best results come from doing all of these steps, in some form or another at some point.

Do research

👤 **User** research: **interviews**, surveys, a/b tests, card sorting, focus groups, and many more

What do you like/dislike? Pain points? Common tasks? Feature wishlist? Watch them use apps.

📈 **Market** research: direct competitors, sites in similar domain, popular sites

Use other apps as a user. What do they do well? What is painful? What ideas can you take?

⚖️ **Document** and **judiciously** incorporate

Users are good at identifying problems, but not solving them

Ref Ref Ref

9

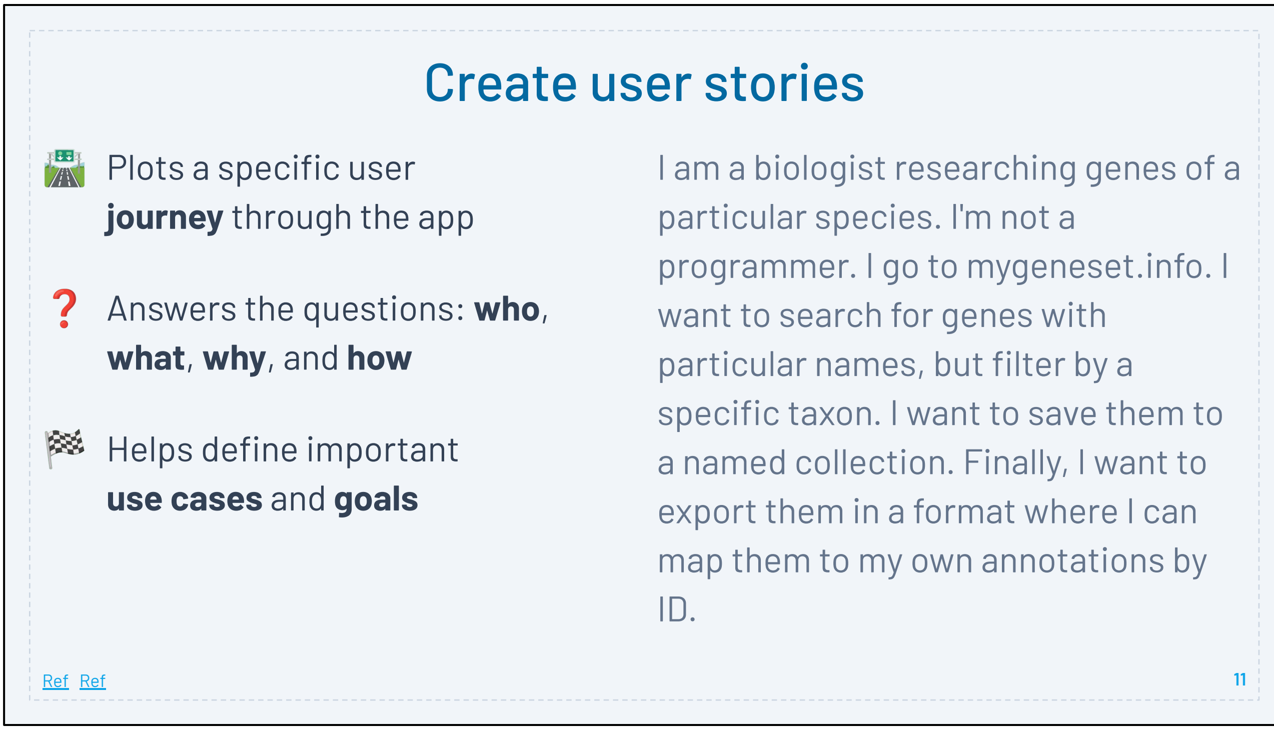
If you can find the time for it, this is one of the most valuable insights you can gain. Ideally, you'd do this at all stages of the process, but realistically it's rare that we have the luxury to do this at all in our case. So I recommend trying to plan a little time for it just in the beginning of the process.

The first big part to this is user testing, which you can do in a plethora of different ways. This is a whole field of study in and of itself, a phrase you'll hear me say several times during this presentation. The one method that I think provides the most value for the effort, and the one I think is most doable by small research groups, is interviews. When developing the new Monarch Initiative site, we conducted something like 20 hours of interviews with a dozen+ people who'd actually be using the app, before even writing a line of code. In the case of Monarch, we were redesigning an existing app, but here are some of the questions just to give you an idea: What do you like/dislike about the current UI? What type of user are you, e.g. clinician, researcher, biologist (getting at target audience and roles)? What are the primary features you think the app should have? How does it compare to other apps you've used? Take us through how you typically use the app, and we'll watch (really valuable). To adapt these questions to a brand new app, you could ask them to imagine a theoretical app, show them a sketch, show them an interactive prototype, or show them an existing app by someone else.

You could also do surveys (standard set of questions, unsupervised), a/b testing (making two versions of a feature and comparing users' response to them), card sorting (putting features/content on cards and asking users to sort them into categories intuitive to them), focus groups (basically group interviewing/brainstorming). And this is just the tip of the iceberg, there's tons more: accessibility testing, tree testing, diaries, analytics, to name a few.

Market research is another important part, and can often be a sobering reality check. Find other apps trying to do the same thing you're trying to do, and use them extensively. Here you have the advantage of being an unbiased third party perspective, i.e. a user. What do they do well, what is frustrating about them? See if you can you can create something that is the best of all worlds. What ideas can you take from them? "Good artists copy, Great artists steal". And this doesn't just apply to scientific apps. If you have doubts about how something on your site works, see if something similar has been done on a popular site, e.g. Amazon, Google, whatever. This is something I did extensively when designing Lab Website Template. I found all the other website templates going after the same niche, and I looked at all the features they had and I didn't, and slowly implemented them.

Finally, you need to be capturing and documenting all of this so you can distill it into a working design. Keep in mind that you don't have to, and probably *shouldn't*, incorporate every single thing your users say. Something I've come to believe, and I've seen people in all sorts of different professions say this, is that users are usually right when they say something is wrong, but are usually wrong about how to fix it. Ultimately it's up to you to decide what to incorporate and *how*. If you said yes to everything, you'd end up with a mess, i.e. design by committee. Something that tries to be good at everything ends up being good at nothing. You need to make opinionated, deliberate, unified, cohesive decisions.



User stories are little vignettes that capture vertical slices of your app. There can be many formats to these, but they typically answer the following questions.

Who/where/when: Defines the audience and particular perspective. In our case, that might be another researcher or a clinician. Or, it could be a lay-person that might need things presented in the most plain language with auto-selected defaults.

What: Defines what action the user wants or tries to take, or a feature that they want. E.g., to generate a report, a shopping cart.

Why: Defines the motivation behind what they want to achieve. This can inform how the feature is implemented. Example: If a report is needed for analysis purposes, it should be exported as a CSV file. If a report is needed for presentation purposes, it should be a PDF.

How: Walks through step by step what the user thinks or expects to do to accomplish their goal. I click on this, then I go to this page and do this, etc. In the most traditional definition, a user story doesn't include this, but it can be very valuable.

Here's a basic example, and you can go much much further and deeper. This one doesn't even cover the how.

Since I've encountered this a lot here, I want to make a note of it: searching. You can split searching into two general categories: exploratory and known-item/re-finding.

Think about how many users will be coming to your site to just poke around and see what the deal is, vs. how many users will have a specific thing in mind that they want to find/re-find. E.g., if you're expecting more exploratory users, you might make sure search is very fuzzy, and if you're expecting more known-item users, you might implement a convenient favorite/bookmark or "most recent searches" feature. These are things you can incorporate right into user stories, e.g. "I am a returning user trying to look at my most interesting gene".

I like user stories because they're fairly easy to do, they force you to focus on the user first and foremost, and they help you define the most important use cases and goals that you can compare your design and implementation against. It could also help clarify what is in scope and what is not.

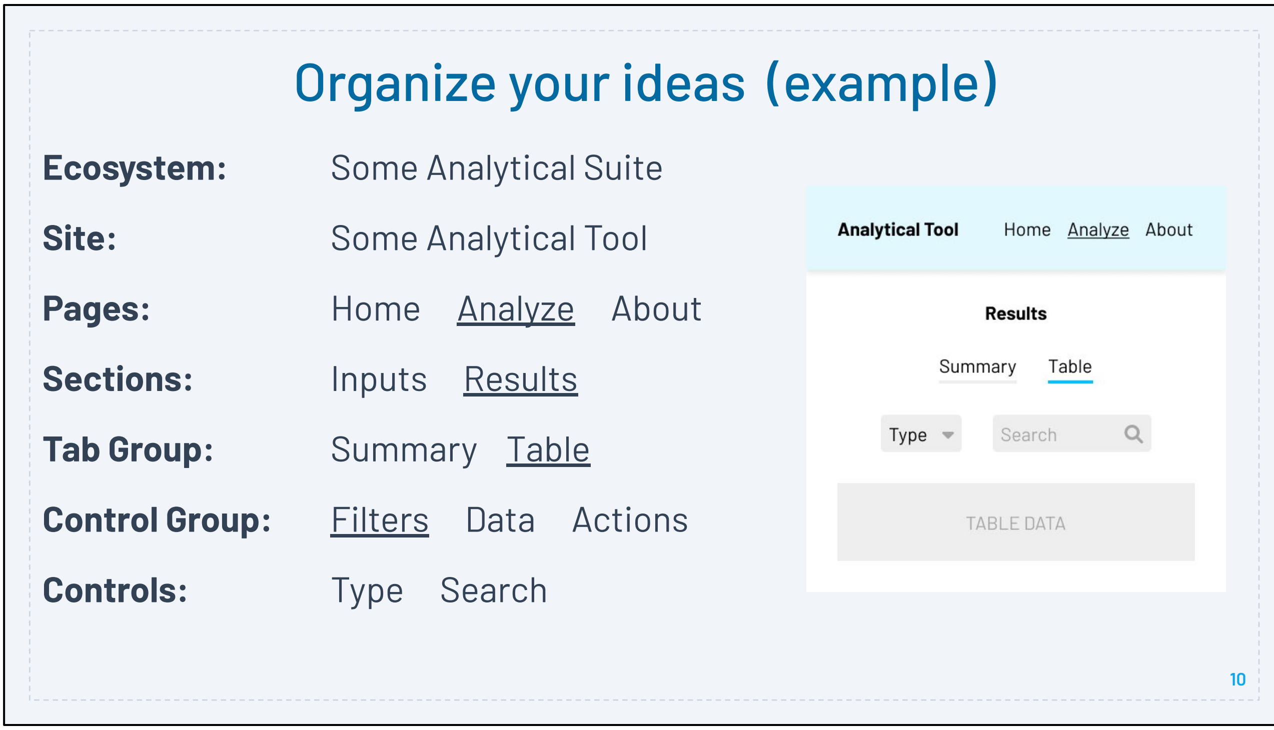
How you categorize and split up ideas has a big impact on how clear and intuitive your app ends up being. If your mind is messy, your app will be too (related: Conway's Law). You might see this referred to as "information architecture", and there are people who specialize in just this. And notice I'm saying ideas, not content, because it's not just about the tangible content on the screen, it's about the concepts underlying it too.

Think about how a user might have a simplified understanding or mental map of your content. Could a layperson get the idea? Luckily a lot of the apps we make are the same, so I can give you a good example: inputs vs. outputs. The user answers some questions or selects some things or uploads some data, then gets information back and can view or download it in various forms. A basic, clear, intuitive split. Another example: a shopping cart; add items then checkout. People would get it instantly. Establishing a model like this naturally leads you to make more intuitive design choices.

Split your content and functionality into intuitive groups, from top to bottom. What are the most important things? What are the features you want to highlight, the "bread and butter", the "special sauce"? Put these front-and-center. This applies at the site level, but at the page and section levels too. We'll see an example on the next slide.

Once you've established models and categories, stick to them. Now of course, software is never completely nailed from the start. You'll end up adding new features and switching things around. When this happens, make sure you ask how the changes fit into the existing model. Does it feel like they're being "tacked on"? Maybe the models have to be expanded or refactored. Just slapping new features in is what leads to ad-hoc, piecemeal designs that look like a messy patchwork of duct-tape to the user. This is something I see a lot. A project is worked on over several years with multiple people coming in and out. This is really important, so just be aware of it and make time for it.

Also, it's okay if you have small pieces of various information that don't fit cleanly into a category. "Miscellaneous", "Details", "More Info", "Extras" can be their own categories. It's okay to have a little bit of entropy if it's labeled as such, and cordoned off. Just consider the size and importance of the information before resorting to this.



On the left are the typical navigational levels of a site. In the middle is how you could organize your ideas into those levels (underlining one item as an example to follow down to the next level). For the order in each level, consider which are most/least important/common.

Some more ideas of how you could split up ideas: first party (our data) vs. third party (their data), filters (affects data) vs. actions (causes side-effect), visual (high-level) vs. tabular (detailed), split by question user is trying to answer. Look for clear, natural divisions.

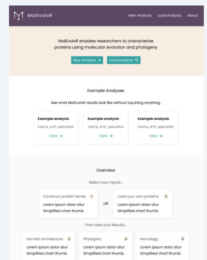
There can be another level above site: ecosystem. It's really hard to make a website that's clear when the overarching collection of sites, products, tiers, tools, versions, etc. is messy. I've encountered this with Manubot and Monarch Initiative for example, where it is not just one thing but a collection of various things that all work together. Try to make the division of these things simple and clear. Make sure you have illustrations of the differences in some central place, and also examine whether you can change the nature of the pieces themselves to make the distinctions better, e.g. this package processes the data, this piece analyzes the data, this website visualizes the data.

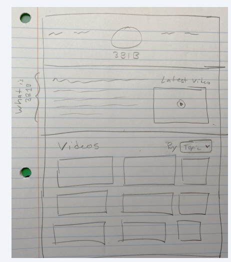
Make a sketch

🖼️ **Design** should **inform** implementation, not the inverse

✏️ **Draw** and **map** your ideas first. Coding comes later.

🛠️ Try Figma, Lucidchart, or pencil & paper



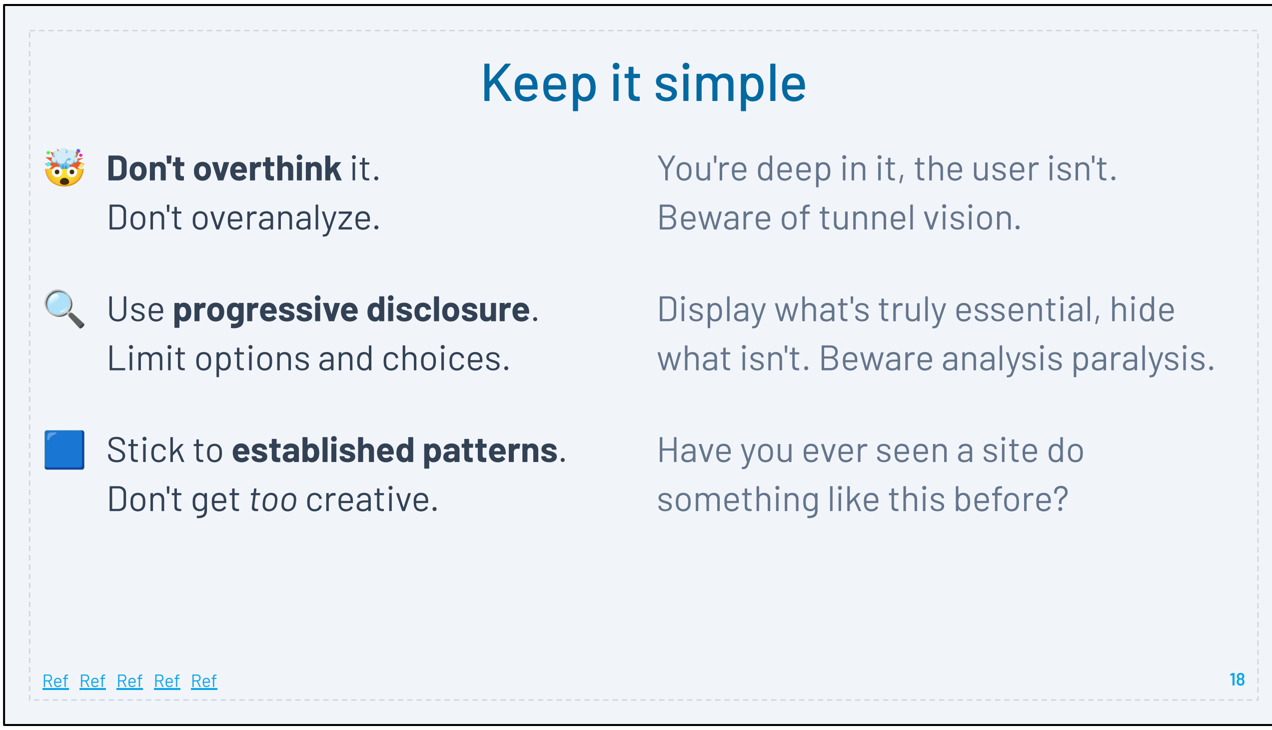


As much as possible, get out of the headspace of a programmer or scientist and into the headspace of a user and designer. Some people use the pejorative term "programmer art" for something that was done too much from a technical perspective and is obtuse or ugly to the average person. The design should inform the code, not the other way around.

To that end, don't start by coding, start by drawing. When you're burdened with thinking about implementation details, you can't focus on the bigger picture. You end up limiting the design by what you think you can or can't implement. Try making a sitemap. Some people call this a "top-down" approach, where you're focused on the bigger picture first, not the finer details.

You can use a full mockup tool like Figma, which let's you establish both the exact aesthetic and the organization at the same time. You can use a flowcharting tool like Lucidchart to just make a basic hierarchy, or you can just quickly sketch it out on paper. You could even make an HTML/CSS mockup with no interactivity.

Here are basic tenets of UX advice that really apply to everything.



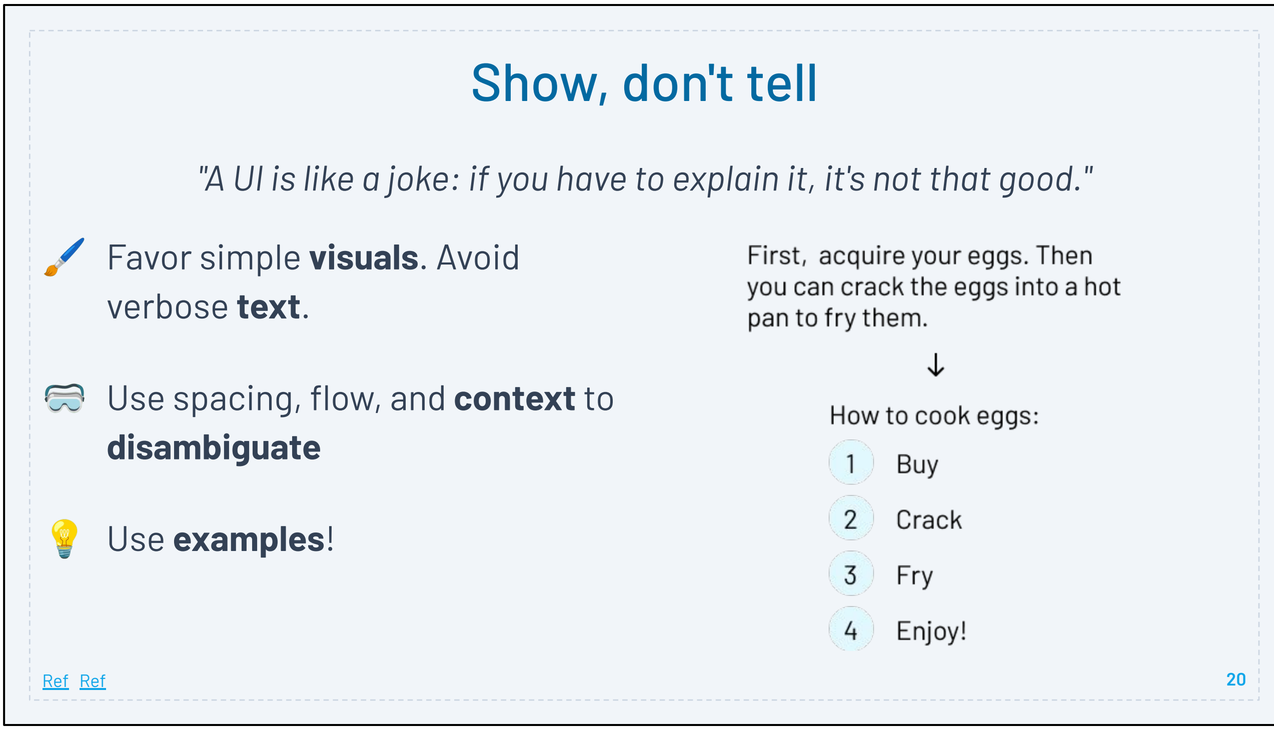
The first part of this is a mindset. When you're an expert in a topic or you're working on a particular project for a very long time, you can get tunnel vision. Sometimes this makes you stare and pick at on a mundane detail that's actually perfectly fine. Other times it makes you assume something is obvious and easy when it's not. You have to remember: for you, this app is your main focus, but for your users, it's just a blip in their day. Try putting yourself in the shoes of a brand new user. What are the gut reactions and impressions? If you can't do that, try stepping away for awhile, or seeking outside perspectives.

Avoid information overload. Show the essential, minimal amount of information first, then allow the user to "drill down" for more details if they need it. Example: Maybe you have a bunch of checkboxes, but 2-3 are the most critical and common. You can hide the rest away under a "show more" button. Another example: you can have a "summary" tab for a quick look, and a "details" tab for those who want a comprehensive report of everything.

You might think that a user should and will always take the time to look at all of your helper text, controls, etc. But again, this is just a small fraction of their day, and their tolerance for that is low. Having too many things on the screen at the same time overwhelms users, reduces comprehension, and frankly makes people just give up. It reduces "scannability" and makes it take longer to find the thing they're looking for.

Another example I see a lot. What needs to be in the header, the thing that is at the top of every page on your site. Does something niche like links to source code *really* have to be accessible from anywhere at any time? Does it really need its own dedicated page? Or can it just go at the bottom of the About page? What % of the time will users really need that? Sure, we want to minimize number of clicks to get to things, but don't obsess over that. It's usually a tradeoff between simplicity and efficiency, and I'd usually pick simplicity.

Lastly, make sure what you're doing has some "precedence". People use websites by applying knowledge and patterns they've learned elsewhere, so don't stray too far outside of that. A button should look like a button, a dropdown should behave like a dropdown, etc. Even academic users, in my experience, use scientific sites as if they were regular websites, even if the functionality and context is completely different.



As much as possible, it should be obvious what a thing does by looking at it, no explanation required. This largely comes from how it looks, which we'll get into this more in the UI section. If you really can't do it visually, simple flowcharts, step lists, or infographics are better than raw text.

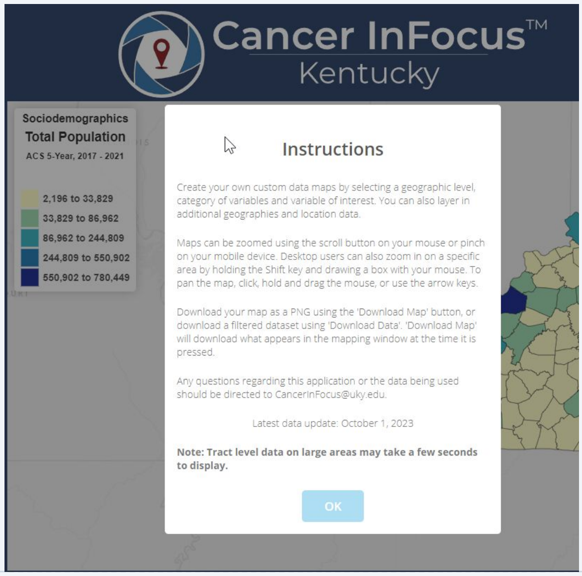
But more critically I think, this comes from a thing's context and flow within the page, i.e. what is it near, what section is it in, what just happened before. A single "download" button right next to an image likely needs no explanation. Notice I don't say "buy eggs, crack eggs, fry eggs", because we're already in the eggs list. It should be unambiguous from the context. Try to construct your UI in a way that each piece has clearly only one thing it could mean.

Now, in research, we deal with a lot of complex topics, so some writing is to be expected. I'm more referring to explaining the UI itself here. But even when explaining scientific things, really consider how much you need to explain with words, or at all. Can you assume your users have some background knowledge in e.g. genomics? At the very least, you can assume they know how to use a website and don't need to be told how to click a button.

Finally, use examples! When I'm learning a new board game, it's way easier for me to just do a practice round than to have someone to rattle off all the rules. Giving basic examples in text or icons or having a button that auto-populates a form is a great way to allow someone to instantly grasp how your app works and explore what it's about. Avoid static screenshots of your UI. Even animated gifs are a form of telling not showing, imo. And it just duplicates your UI, and requires you to re-record anytime there are changes. We have gifs on the homepage of Monarch, but those are more for advertising what features the app has.

In the same vein, in our work we tend to instinctively reach for tables to display data, but think about better ways to summarize and visualize the data, like interactive charts, networks, and the like.

Show, don't tell (cont.)



cancerinfocus.uky.edu

Here's an example of what not to do. Presently, the site gives a text warning about loading, instead of just providing a loading spinner which is universally understood. A popup explains how to use the map, what the download button does, etc. I'd wager that almost all of us have used Google Maps or something similar at some point. Do they explain how to scroll and zoom? No. It's okay to assume some base level of web competency. It's just not practical to explain every single thing.

I think most people neglect/overlook this, and don't even realize this is a part of UX. If you're unaware, copywriting is just the text content of your site.

Be ruthless about cutting words from your help text, alert text, tooltip text, etc. I think researchers are used to writing papers where you use maximal and precise wording. That doesn't translate well to the web and isn't necessary. Instead, what is the bare minimum you can get away with? Don't necessarily have to use "shortened grammar", but do go through every sentence and think about what words are truly necessary. E.g. "small text size" → "small text", small is already a size. I spend a ton of time on this in my apps. People's attention span and memory is limited, as is the physical space on the page.

Use consistent sentence structure. Examples of some consistent patterns you might want to establish: All FAQ's written as questions from the first-person perspective. All help text written in active voice. All buttons written as commands/verbs. All helper text written to same level of detail, 1 or 2 sentences. Also be consistent with your terms. Is it a "job" or a "run" or an "analysis"? Is there a distinction, and if so where/how is this clarified? Users pick up on how and where you use certain language and draw conclusions from that, sometimes subconsciously, so take each word usage seriously.

Use simple words. Get out of the habit of using big words. Example: a "retrieve analysis" button could just be "load/open analysis". Could even get away with just "load/open" if it's clear from the context. "Load" is a basic function of almost all software that everyone understands. "Retrieve" perhaps gives a feeling of being more accurate, but it's really just obscuring the matter. We don't call finding a file on your hard drive "retrieving" or "reading" or "parsing", we simply say "loading".

Avoid idioms. Think about how an automatic translator might misconstrue what you've written.

Overall, just write like you would normally talk to a person. Don't overthink it. Go with your gut and trust that people will understand.

UI Principles

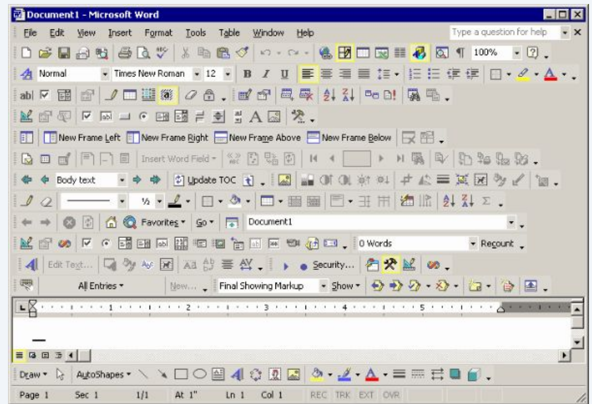
Now for some UI tenets.

Layout is key

💨 Let it breathe! Blank space is your **friend**.

🌊 Keep the **flow** simple and clear. Embrace linearity. **One thing** at a time.

📱 Not all **screens** are desktops! Responsive design.



Ref Ref Ref Image from codinghorror.com

Don't fear blank space. It is your friend not your enemy. I think this is a holdover from make compact, dense papers. When there's too much info, and the brain just glazes over things. Yes, papers are dense and people can read them just fine, but it's a different expectation on the web. Plus, papers are not interactive and usually less visually complex (web has text, tables, images like papers, but also checkboxes, buttons, navbars, menus, popups, etc).

Somehow, you need to make sure your flow is simple and clear. What goes in what order, and what affects what. Imo, not enough people embrace linear, top-down flow. I use this whenever possible because it's a standard paradigm of the web (you scroll down for more), and because it makes order and causation pretty clear. First this happens, then this, then this; this choice affects the things below it, not some other random place. Consider: Do two things really need to be side to side, or do you just want that out of space-saving habit? I'd argue that they only need to be side to side if the user really needs to cross reference both the things back and forth, repeatedly. If it's a thing you just set/choose once or infrequently, it's okay to just put it above and let the user **focus on one thing at a time**. Again this is often a tradeoff between efficiency and simplicity.

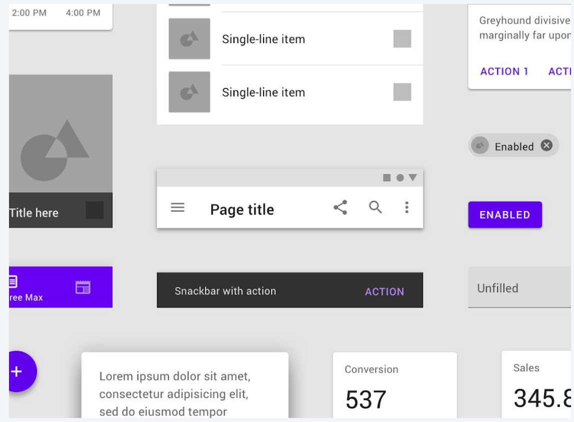
Specifically, section elements should always be stacked vertically (because this is what almost all sites do) and should be demarcated clearly with a background color change or a horizontal rule or a heading (not a header).

One specific comment I get a lot is, "we have all this blank space on the sides, why aren't we using it?". These are the reasons, but also, think about readable content width. You don't want to be darting your eyes a long distance across the screen on every line wrap or every row of a table. This is actually an accessibility issue as well.

You might think scientists are always going to be working on a full sized screen, but I guarantee you that's not always true. Maybe you're at a conference and want to show a colleague/connection your new site, so you pull out your phone, but whoops all your buttons and text are overlapping and nothing's readable. Maybe you're in your hotel room and want to play around with the app on your tablet while in bed. Don't focus on the exact dimensions of the machine you usually work on, like the way a particular paragraph wraps; it's a wild goose chase. There's a practically infinite variety of screen sizes. Focus on the overall look as you resize your browser window around. Note that a typical way to keep things usable on mobile (small screen widths) is to collapse things down to a single column (another reason to embrace linearity). Don't forget landscape on mobile too! This stuff is what is called "responsive design", and it's really not optional nowadays. Your site has to work and look good on mobile.

Establish a visual language

- 🔣 Icons, colors, shapes, fonts, weights, sizes, order, position, spacing, language, etc.
- 🎨 Make a **palette** to enforce consistency
- 🚫 Keep in mind **common associations** and expectations



Ref Ref Ref Image from Material Design (material.io)

What does this include? These things. Positioning (e.g. download button always goes to the bottom right of a chart). Language (like we talked about with copywriting before). Over time using your app, users can be "trained" to expect certain things. Even if they're only visiting for a short time, they can pick up on and deduce meaning. This only works if things are consistent and clear and simple enough though.

We can create what's called a "design system" to decide on and enforce this language. In our case, this can just be a "palette", or a collection of things like icons, colors, fonts, etc. to pick from. For example, you can use a comprehensive icon library like Font Awesome to make sure all your icons look the same. Instead of writing color hex codes everywhere in your code, you can write a handful of them once, as variables (like "accent" or "theme" or "dark"), and then use the variables everywhere. Try to have a standard for spacing and sizes too, like multiples of 4 or 10; don't do it arbitrary/"willy nilly". Be careful about what and how many HTML tags you're wrapping things in (see: "div hell") and their accumulative margins/paddings.

You can make a suite of "components", which are just building blocks for things like buttons, sliders, dropdowns, etc. Then you make sure to always pick from them instead of recreating the wheel *slightly differently* each time. These don't just have to be controls, they can be things like a section or a heading. Or one of my favorites: a "time ago" component that takes a UTC datetime and displays it in a relative form, like "~20 min ago" (with the absolute, full date available on hover, for progressive disclosure). Take a look at [Material UI]() or [Vuetify](), the largest libraries of components for React and Vue, to see what types of industry-standard components exist. You can also find "headless" component libraries where it's easy to apply your own styles and make them fit right into your aesthetic.

And don't just pick a component at random! Say you have set of radio buttons with 12 different options. Does the user really need to see all of those options at the same time? That can be a dropdown select instead. Progressive disclosure.

A nice side effect of all of this, too, is you end up establishing a brand identity. Your site becomes more memorable and distinguishable from the countless others. People will actually enjoy using it.

Finally, make sure you keep in mind common associations and expectations. These can be very culturally dependent, but for example, red is often used to indicate an error and green success.

Accessibility

This is really its own category all together, but I have a moral obligation to talk to you about it here, because it's almost always ignored and a lot of people aren't even aware of it. There are people who dedicate their careers to just this.

What is accessibility?

Partial vision disabilities

Color blind (7+ different kinds!)

Motor disabilities

Cognitive disabilities

Hard of hearing

Communication disorders

Psychological disabilities

Motion sickness

Seizure disorder

Physical pain

Respect accessibility. Not everyone is like you.

Ref Ref Ref

So what is accessibility, it's just referring to people who are blind right? No, not at all. Time for a class exercise. Try to think of as many cases or types of people we might need to consider to make sure our site is inclusive and can be used by everyone.

These can be split in various different ways. This is also still a nascent field, and I think that even the experts are not experts. We have a long way to go with this. So the main takeaway here is never be confident that you know all of the challenges your users face.

Common accessibility issues

⌨️ No keyboard focus

📃 No alt-text, CC

😶 Non-semantic HTML

⚓ No headings

👁️‍🗨️ Bad screen reader support

🔎 Low contrast, small text

🌈 Color-reliant

👆 Difficult click targets

💫 Convoluted language

💥 Flashing, motion

Use **Lighthouse** (in Chrome dev tools) or **Deque Axe** directly

Ref Ref Ref

Here are some common accessibility issues you need to be cautious of all the time.

Making things reliant on a mouse. Many people use computers with the keyboard only, even abled people. Sometimes I'm eating a sandwich with one hand and want to continue using a site with just the keyboard. Make sure interactive elements can be focused by tabbing to them.

Hopefully most of you are familiar with screen readers. These are programs that help people navigate and read websites by announcing content. Make sure images, charts, and other visual elements have some kind of caption or descriptive text that can be accessed by SRs (unless they're solely decorative). This is really a whole art-form itself, so look up tips on how to do it. Relatedly, if you have a video on your site, make sure it has closed captions for people who are hard of hearing.

This is getting more technical, but the type of HTML tag you use is critical. Use <a> for navigating somewhere, <button> for performing an action, <p> for a paragraph. For every tag you write, look up whether there's a defined meaningful HTML tag that's appropriate for your use case. <div> and <span> have no meaning on their own.

Use headings! SR users use headings (not headERs) to jump around and navigate the content on the page, like a table of contents. People don't use these enough. They can be put on a page "invisibly", just for SR users, if you feel your visual layout makes things clear enough for fully sighted users.

In general, just keep screen readers in mind. And actually try navigating your site with them. On MacOS, press command F5 to activate VoiceOver. You'll learn something.

Some more specific issues:

Maybe someone is hard of sight and needs high contrast and large text. Don't use bright yellow on a white background. Don't use font-sizes much smaller than the document root font size. As a quick and dirty check, I sometimes squint my eyes and pull back from my screen to see if things are still legible.

Maybe someone is color blind and can't see the difference between green success text and red error text. You can use color, of course, but make sure essential things don't *rely* on it.

Maybe someone has only the use of one hand, or lacks of fine motor control or dexterity, and so needs click targets to be larger and spaced more apart.

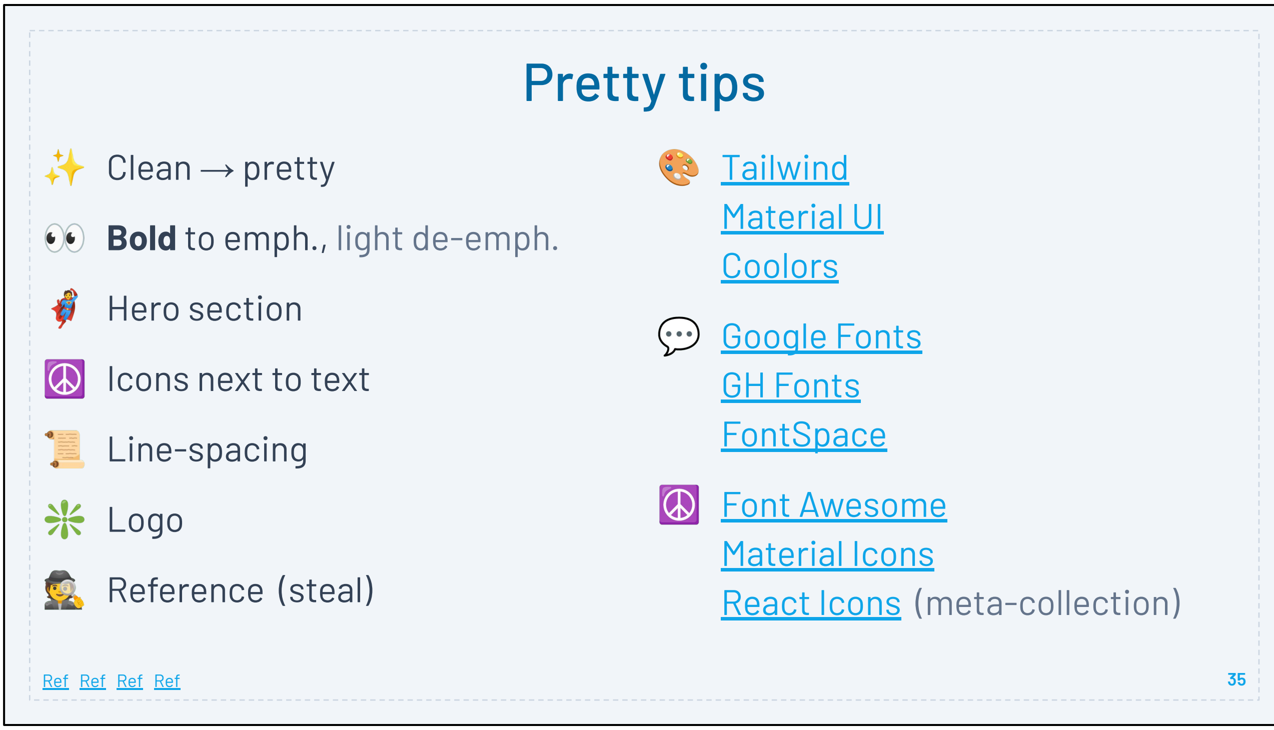
Maybe someone didn't have access to education or has a learning disability and can't read at a collegiate level, or maybe English is not their first language. Don't gatekeep your site from them by using convoluted language.

Maybe someone has seizure disorder or motion sickness, and too much animation will cause them physical harm or discomfort.

This is all very difficult to get right, and frankly I think not even experienced developers get it right most of the time. Luckily we have some good tools to help us with this. Lighthouse, a tab in the Chrome dev tools panel, will check the page for all the accessibility issues it can. You can also use Deque Axe (what Lighthouse uses under the hood) directly, or set up automated accessibility checks. We do this in Monarch, it's an actual test case that has to pass before a PR can be merged. Just keep in mind that it can't catch everything. Accessibility testing with actual users is still the golden standard. Also sometimes there are false positives, and there is some disagreement around color contrast calculations. But it's a super useful tool that catches a lot of issues.

Tips

Now for some smaller, more tangible tips.



A lot of making things pretty comes down to just experience and practice, but here are some tips.

Again, don't overthink it. Prettiness will come from cleanness. Keep it simple.

You can use bold words to draw users' eyes to the most important keywords. And you can use light gray text to show "secondary" or less important information (but not too light, remember accessibility). You may have noticed I've been doing that in these slides.

A lot of sites have what's called a hero section on their homepage, which is usually just a full screen width, eye grabbing section with a highlight, visual, and/or very short text. See Apple's current site. On Simplex for Arjun, it has a (very) brief explanation of what the site is and why you should care.

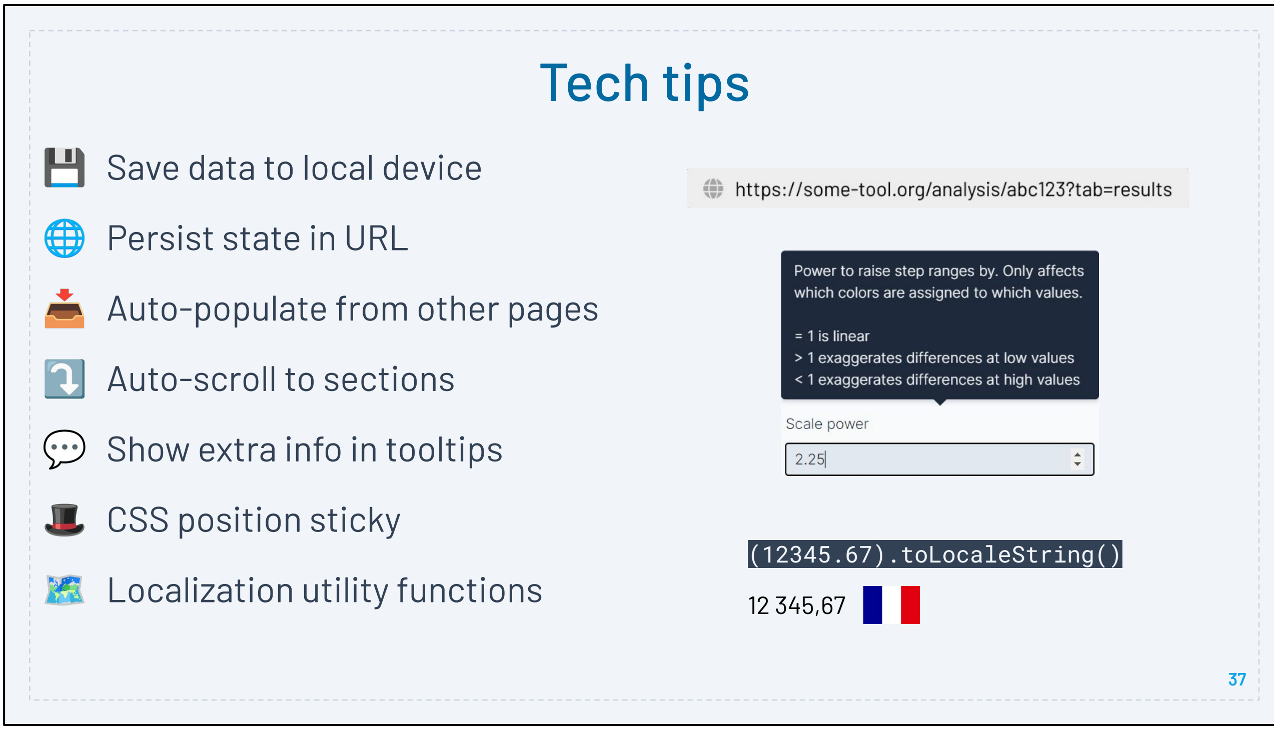
Use an icon next to a menu item or button text. Not only does this look nicer, but it actually let's users more quickly recognize and find items. But don't use *just* an icon too often. If you do, the meaning has to be absolutely clear, and you need to make sure it has an accessible hidden label too.

This is the "let it breathe" principle, applied to text. I notice a lot of people stick with default single spaced paragraphs. Try opening that up to 1.5 or even 2. It tends to look more pro and be more readable, especially with wider paragraphs.

Try to make a nice logo. This could be a whole course itself, and I'm not really a logo designer, but here are some quick tips. Don't make it too busy. Don't shove too much in there. It has to look good and be legible at favicon (bookmark bar icon) size. It can't possibly capture entirely what your site is, so don't obsess over the meaning. Does Apple's logo capture what its company is? Not even a little. Write down some keywords about your app and synonyms from a thesaurus, then look them up on Google or the Noun Project for visual ideas and play around with combining them.

Lastly, look at other cool websites! See what looks cool and adapt it for your use case. Start bookmarking cool designs when you see them.

Here are some resources I constantly use. Professionally designed color palettes, fonts, and icons, curated to be cohesive and work well together.



I said we wouldn't go into technical things in this presentation, but there are a few things you need to be aware are *possible*, because they can lead to great design simplifications.

You can save limited data to the user's local device (phone, laptop, whatever) with localStorage and sessionStorage; up to about 5 MB. You could use this to save the user's recent searches or most visited items. Or you could save the analysis IDs the user submitted so you don't have to make a login system.

You can put limited state in the URL, up to about 1,000 characters. This can include things like the current analysis, selected options, selected tab, etc. This doubles as a "share" feature, because your users can just copy the link in the address bar and paste it to someone, a thing people know how to do from e.g. Google Maps (url contains coordinates, zoom level, etc.).

You can auto-populate things from other pages, i.e. send data from one page to another. This can be done in various ways: with the router, with the aforementioned storages, in-memory, etc. Example: in Gene Plexus, we plan to allow the user to "edit" the options for an analysis that was already completed by having a "duplicate and edit" button that just copies the values over to the "new analysis" page and lets the user edit them from there.

You can auto-scroll to sections. Not everything has to be in tabs and right at the top of the page. Things can be vertically stacked. You can also have a table of contents on the side.

You can have tooltips, which are boxes of (ideally minimal) content that appear when hovering over or focusing something. In the spirit of progressive disclosure, you can display just the absolute bare basics on the page, then have a bit more explanation in a tooltip. You should still keep this content as minimal as possible though.

CSS has a feature that allows an element to "stick" to one of the sides of the viewport (usually the top) as the user scrolls. This can keep something always in view, if it's critical or needs to be changed frequently. You could make the page's header sticky, or maybe just the name of the analysis you're on. This is like frozen rows/cols in Excel.

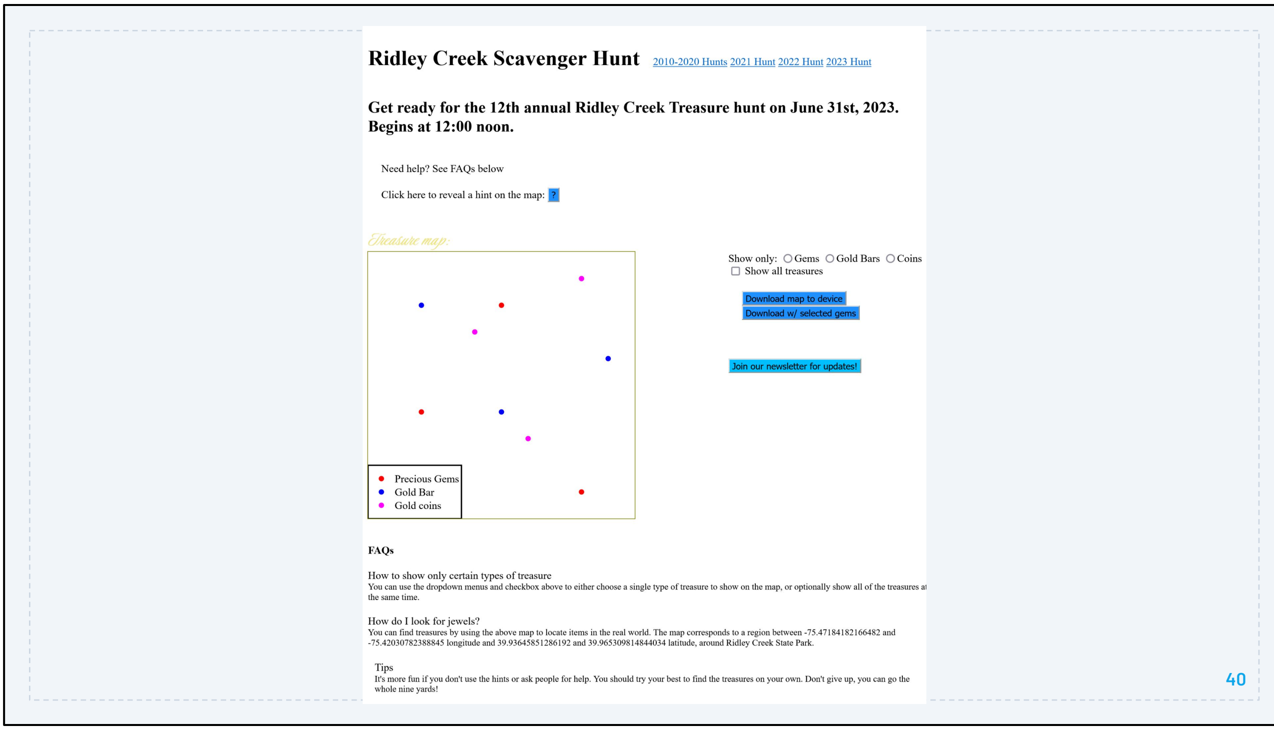
There are some functions that can automatically format dates, times, numbers, and things like that depending on the country the user is in. For example, some countries switch the roles of commas and periods in numbers (from what we in the US are used to).

Exercise

Related example of bad UX

Some issues with related example:

- Explanation at top ("to make a search…") shouldn't be necessary. Some of it is also repeated below it.
- In "Type A Place Name", "type a" is not necessary (it's obviously a text box), and "place name" is already said in the label above it. Should be an example, like "ex. London, UK".
- "For better accuracy" sentence belongs right near the "place name" input, not separated by unrelated things, and could be reworded to "Use name, address, city, state, zip for better accuracy".
- Lat and long inputs are not really inputs! They're input HTML elements, but typing in them does nothing. They're really outputs, and you have to click on the map to get them to change. This is explained at the top, but shouldn't be necessary.
- Facebook and Twitter buttons should be down with the other share stuff, not randomly in the green input box.
- Repeated text everywhere.



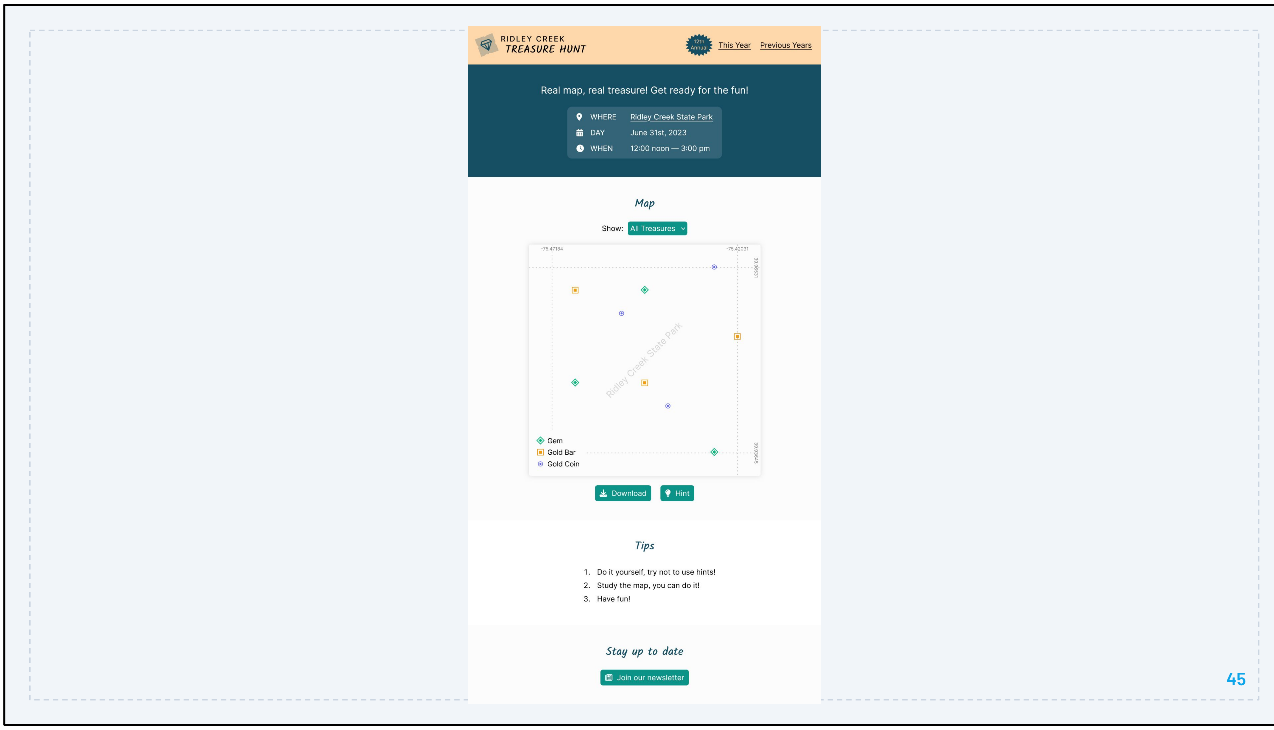

```html
<html>
  <head>
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link
      href="https://fonts.googleapis.com/css2?family=Luxurious+Script&display=swap"
      rel="stylesheet"
    />
  </head>
  <body>
    <header>
      <h1 style="display: inline-block">Ridley Creek Scavenger Hunt</h1>
         
      <div style="display: inline-block">
        <a href="2010">2010-2020 Hunts</a>
        <a href="2021">2021 Hunt</a>
        <a href="2022">2022 Hunt</a>
        <a href="2023">2023 Hunt</a>
      </div>
    </header>
    <h2>
      Get ready for the 12th annual Ridley Creek Treasure hunt on June 31st,
      2023. Begins at 12:00 noon.
    </h2>
```

```
<div style="padding: 20px">
  <div>Need help? See FAQs below</div>
  <br />
  Click here to reveal a hint on the map:
  <button style="background: dodgerblue">?</button>
</div>
<br />
<b style="color: khaki; font-family: 'Luxurious Script'; font-size: 30px"
  >Treasure map:</b
>
<div>
  <svg
    viewBox="-50 -50 100 100"
    style="width: 400px; outline: solid 1px olive"
  >
    <circle fill="red" r="1" cx="0" cy="-30" />
    <circle fill="red" r="1" cx="30" cy="40" />
    <circle fill="red" r="1" cx="-30" cy="10" />
    <circle fill="blue" r="1" cx="40" cy="-10" />
    <circle fill="blue" r="1" cx="0" cy="10" />
    <circle fill="blue" r="1" cx="-30" cy="-30" />
    <circle fill="magenta" r="1" cx="10" cy="20" />
    <circle fill="magenta" r="1" cx="-10" cy="-20" />
    <circle fill="magenta" r="1" cx="30" cy="-40" />
    <rect
      fill="none"
      stroke="black"
      stroke-width="0.5"
      x="-50"
      y="30"
      width="35"
      height="20"
    />
    <circle fill="red" r="1" cx="-45" cy="35" />
    <circle fill="blue" r="1" cx="-45" cy="40" />
    <circle fill="magenta" r="1" cx="-45" cy="45" />
    <g style="font-size: 4" dominant-baseline="middle">
      <text x="-40" y="35.5">Precious Gems</text>
      <text x="-40" y="40.5">Gold Bar</text>
      <text x="-40" y="45.5">Gold coins</text>
    </g>
  </svg>
  <div style="float: right; width: 300px">
    Show only:
    <input name="filter" type="radio" value="Gems" />Gems
```

```
      <input name="filter" type="radio" value="Gold bars" />Gold Bars
      <input name="filter" type="radio" value="Coins" />Coins
      <input type="checkbox" /> Show all treasures
      <br />
      <div style="padding: 20px">
        <button style="background: dodgerblue">Download map to device</button>
        <button style="background: dodgerblue">Download w/ selected gems</button>
      </div>
      <br />
      <br />
      <button style="background: deepskyblue">
        Join our newsletter for updates!
      </button>
    </div>
  </div>
  <br />
  <br />
  <b>FAQs</b>
  <br />
  <br />
  <div>How to show only certain types of treasure</div>
  <small
    >You can use the dropdown menus and checkbox above to either choose a
    single type of treasure to show on the map, or optionally show all of the
    treasures at the same time.</small
  >
  <br />
  <br />
  <div>How do I look for jewels?</div>
  <small
    >You can find treasures by using the above map to locate items in the real
    world. The map corresponds to a region between -75.47184182166482 and
    -75.42030782388845 longitude and 39.93645851286192 and
39.965309814844034 latitude,
    around Ridley Creek State Park.</small
  >
  <br />
  <div style="padding: 20px 10px">
    <div>Tips</div>
    <small
      >It's more fun if you don't use the hints or ask people for help. You
      should try your best to find the treasures on your own. Don't give up,
      you can go the whole nine yards!</small
    >
  </div>
```

```
  </body>
  <!--
    ISSUES:
    - Ugly, obviously. Default browser styles.
    - Header nav can be reduced to "previous years" and "this year".
    - Date and time could be dynamically generated with JS auto-locale, however this
is a local event and it's probably not necessary. Know your audience.
    - "Get ready" text could be nice looking hero section.
    - Doesn't show most important info first.
    - Show date and time as nice mini-table of "Where:" and "When:", with icons.
    - Padding and spacing is all over the place.
    - Some sentences don't have closing period.
    - Hint button should be next to map with other map controls, and should just say
"Hint" with ? icon.
    - Buttons slightly different shades of blue
    - Treasure map font and color illegible.
    - Inconsistent terms for treasures, gems, bar vs. bars, etc.
    - Filters can be reduced to a single select dropdown.
    - Download map button is too wordy, and should go next to reveal hint button
because it's an action.
    - Second download map button is unnecessary. Just have one download button,
and have it download whatever are the currently selected options.
    - Newsletter button doesn't apply to map specifically, but is next to it.
    - On smaller screens, things get cut off.
    - Symbols on map are all circles, and red and magenta are close together colors.
    - FAQs way too wordy.
    - FAQs have different sentence structures.
    - In fact, FAQs are not really needed at all.
    - Lat/long coords can just go on map. Show, don't tell.
    - Tips section lumped in with FAQs, and has an idiom.
    - Host of Lighthouse errors.
  -->
</html>
```

Exercise (cleaned up)




```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,initial-scale=1.0" />
    <title>Ridley Creek Treasure Hunt</title>
    <meta
      name="description"
      content="The 12th annual treasure hunt at Ridley Creek State Park!"
    />
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link

href="https://fonts.googleapis.com/css2?family=Inter&family=Kalam&display=swap"
      rel="stylesheet"
    />
    <link
      href="https://use.fontawesome.com/releases/v6.4.0/css/all.css"
      rel="stylesheet"
    />
    <style>
      :root {
        /* right on the line of wcag color contrast calc. OK by inspection. */
```

```css
  --theme: color-mix(in hsl, #0d9488 100%, #0f766e);
  --dark: #164e63;
  --light: #fed7aa;
  --rounded: 5px;
}
body {
  margin: 0;
  font-family: "Inter", sans-serif;
  line-height: 1.5;
}
header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  flex-wrap: wrap;
  gap: 20px;
  padding: 20px;
  background: var(--light);
}
@media (max-width: 600px) {
  header {
    flex-direction: column;
    justify-content: center;
  }
}
h1 {
  display: inline-grid;
  grid-template-columns: auto auto;
  grid-template-rows: auto auto;
  align-items: center;
  gap: 5px 10px;
  grid-template-areas: "logo title" "logo subtitle";
  margin: 0;
  line-height: 1;
}
@media (max-width: 250px) {
  h1 {
    justify-items: center;
    grid-template-columns: auto;
    grid-template-areas: "logo" "title" "subtitle";
    text-align: center;
  }
  .subtitle {
    padding: 0 !important;
  }
```

```
}
.logo {
  grid-area: logo;
  transform: rotate(-20deg);
  height: 40px;
}
.title {
  grid-area: title;
  font-size: 1rem;
  text-transform: uppercase;
  letter-spacing: 2px;
}
.subtitle {
  grid-area: subtitle;
  padding-left: 10px;
  font-family: "Kalam", cursive;
  font-size: 1.4rem;
  text-transform: uppercase;
}
nav {
  display: flex;
  align-items: center;
  flex-wrap: wrap;
  gap: 10px 20px;
}
@media (max-width: 350px) {
  nav {
    flex-direction: column;
  }
}
svg text {
  text-anchor: middle;
  dominant-baseline: middle;
}
.annual {
  height: 1.7em;
  overflow: visible;
}
.annual path {
  fill: var(--dark);
  transform: scale(1.1, 0.9);
}
.annual text {
  font-size: 1.3em;
  letter-spacing: 0;
```

```css
  fill: var(--light);
}
a {
  color: currentColor;
}
a:hover {
  opacity: 0.75;
}
section {
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 20px;
  padding: 40px max(40px, (100% - 1000px) / 2);
  background: white;
}
section:nth-child(even) {
  background: #fcfcfc;
}
.hero {
  background: var(--dark);
  color: white;
}
.hero-text {
  font-size: 1.3rem;
  line-height: inherit;
}
.hero-details {
  display: inline-grid;
  grid-template-columns: 20px auto auto;
  align-items: center;
  gap: 10px;
  padding: 10px 20px;
  background: #ffffff20;
  border-radius: 10px;
}
.hero-details :nth-child(3n + 2) {
  text-transform: uppercase;
  padding-right: 20px;
}
@media (max-width: 420px) {
  .hero-details {
    grid-template-columns: 20px auto;
    row-gap: 5px;
  }
```

```css
  .hero-details :nth-child(3n + 3) {
    grid-column: span 2;
    padding-bottom: 15px;
  }
  .hero-details :last-child {
    padding-bottom: 0;
  }
}
h2 {
  margin: 0;
  font-size: 1.5rem;
  font-family: "Kalam", cursive;
  color: var(--dark);
}
label {
  display: flex;
  align-items: center;
  gap: 10px;
}
select {
  border: none;
  border-radius: var(--rounded);
  padding: 5px;
  background: var(--theme);
  color: white;
  font: inherit;
  cursor: pointer;
}
select:hover {
  background: var(--dark);
}
.map {
  box-shadow: 0 0 10px #00000020;
  border-radius: var(--rounded);
  max-width: 500px;
}
.map-area {
  font-size: 4px;
  transform: rotate(-45deg);
  opacity: 0.15;
}
.graticules text {
  font-size: 2px;
  stroke: white;
  stroke-width: 0.5;
```

```
  paint-order: stroke;
  opacity: 0.5;
}
.graticules path {
  fill: none;
  stroke: black;
  stroke-width: 0.25;
  stroke-dasharray: 0.5 1;
  opacity: 0.25;
}
.legend-text text {
  font-size: 3px;
  text-anchor: start;
}
.gem {
  fill: #10b981;
  font-size: 5px;
}
.bar {
  fill: #f59e0b;
  font-size: 4px;
}
.coin {
  fill: #6366f1;
  font-size: 3.5px;
}
.actions {
  display: flex;
  justify-content: center;
  gap: 20px;
  flex-wrap: wrap;
}
p {
  margin: 0;
  line-height: 2;
}
button {
  display: flex;
  align-items: center;
  gap: 10px;
  border: none;
  border-radius: var(--rounded);
  padding: 5px 10px;
  background: var(--theme);
  color: white;
```

```
      font: inherit;
      cursor: pointer;
    }
    button:hover {
      background: var(--dark);
    }
    ol {
      margin: 0;
    }
    li {
      margin: 5px 0;
      padding-left: 10px;
    }
  </style>
</head>
<body>
  <header>
    <h1>
      <svg viewBox="-50 -50 100 100" class="logo">
        <path
          fill="var(--dark)"
          opacity="0.25"
          d="M -50 -50 L 0 -50 L 5 -45 L 20 -50 L 50 -50 L 50 0 L 45 5 L 50 20 L 50 50 L
10 50 L 0 45 L -10 50 L -20 45 L -30 50 L -50 50 L -50 0 L -45 -5 L -50 -20            "
        />
        <path
          fill="none"
          stroke="var(--dark)"
          stroke-width="5"
          stroke-linejoin="round"
          d="M -30 -10 L 0 30 L 30 -10 L 20 -25 L -20 -25 z M -30 -10 L 30 -10 M 0 30 L
-10 -10 M 0 30 L 10 -10            "
        />
      </svg>
      <div class="title">Ridley Creek</div>
      <div class="subtitle">Treasure Hunt</div>
    </h1>
    <nav>
      <svg viewBox="-40 -25 80 50" class="annual">
        <path
          d="M 0.0 -50.0 L 6.3 -39.5 L 15.5 -47.6 L 18.2 -35.6 L 29.4 -40.5 L 28.3 -28.3 L
40.5 -29.4 L 35.6 -18.2 L 47.6 -15.5 L 39.5 -6.3 L 50.0 0.0 L 39.5 6.3 L 47.6 15.5 L
35.6 18.2 L 40.5 29.4 L 28.3 28.3 L 29.4 40.5 L 18.2 35.6 L 15.5 47.6 L 6.3 39.5 L 0.0
50.0 L -6.3 39.5 L -15.5 47.6 L -18.2 35.6 L -29.4 40.5 L -28.3 28.3 L -40.5 29.4 L
-35.6 18.2 L -47.6 15.5 L -39.5 6.3 L -50.0 0.0 L -39.5 -6.3 L -47.6 -15.5 L -35.6 -18.2
```

```
L -40.5 -29.4 L -28.3 -28.3 L -29.4 -40.5 L -18.2 -35.6 L -15.5 -47.6 L -6.3 -39.5 z"
          />
          <text y="-12">12th</text>
          <text y="12">Annual</text>
        </svg>
        <a href="2023">This Year</a> <a href="prev">Previous Years</a>
      </nav>
    </header>
    <main>
      <section class="hero">
        <p class="hero-text">Real map, real treasure! Get ready for the fun!</p>
        <div class="hero-details">
          <i class="fa-solid fa-location-dot"></i>
          <div>Where</div>
          <div><a href="link-to-google-maps">Ridley Creek State Park</a></div>
          <i class="fa-solid fa-calendar-days"></i>
          <div>Day</div>
          <div>June 31st, 2023</div>
          <i class="fa-solid fa-clock"></i>
          <div>When</div>
          <div>12:00 noon &mdash; 3:00 pm</div>
        </div>
      </section>
      <section>
        <h2>Map</h2>
        <label>
          Show:
          <select>
            <option>All Treasures</option>
            <option>Gems ◈</option>
            <option>Gold Bars ▣</option>
            <option>Gold Coins ⦿</option>
          </select>
        </label>
        <svg
          viewBox="-50 -50 100 100"
          class="map"
          aria-label="Treasure map. Contact jane@gmail.com for assistance."
          role="img"
        >
          <text class="map-area">Ridley Creek State Park</text>
          <g class="graticules">
            <path
              d="M -40 -50 L -40 30 M 40 -50 L 40 50 M -50 -40 L 50 -40 M -25 40 L 50 40"
            />
```

```
      <text x="-40" y="-48">-75.47184</text>
      <text x="40" y="-48">-75.42031</text>
      <text transform="translate(48, -40) rotate(90)">39.96531</text>
      <text transform="translate(48, 40) rotate(90)">39.93645</text>
    </g>
    <g class="gem">
      <text x="0" y="-30">◈</text>
      <text x="30" y="40">◈</text>
      <text x="-30" y="10">◈</text>
    </g>
    <g class="bar">
      <text x="40" y="-10">▣</text>
      <text x="0" y="10">▣</text>
      <text x="-30" y="-30">▣</text>
    </g>
    <g class="coin">
      <text x="10" y="20">⦿</text>
      <text x="-10" y="-20">⦿</text>
      <text x="30" y="-40">⦿</text>
    </g>
    <g>
      <text x="-45" y="35" class="gem">◈</text>
      <text x="-45" y="40" class="bar">▣</text>
      <text x="-45" y="45" class="coin">⦿</text>
    </g>
    <g class="legend-text">
      <text x="-42" y="35">Gem</text>
      <text x="-42" y="40">Gold Bar</text>
      <text x="-42" y="45">Gold Coin</text>
    </g>
  </svg>
  <div class="actions">
    <button data-tooltip="Download map image">
      <i class="fa-solid fa-download"></i>Download
    </button>
    <button
      data-tooltip="Reveal a path to one of the treasures on the map"
    >
      <i class="fa-solid fa-lightbulb"></i>Hint
    </button>
  </div>
</section>
<section>
  <h2>Tips</h2>
  <ol>
```

```html
        <li>Do it yourself, try not to use hints!</li>
        <li>Study the map, you can do it!</li>
        <li>Have fun!</li>
      </ol>
    </section>
    <section>
      <h2>Stay up to date</h2>
      <button>
        <i class="fa-regular fa-newspaper"></i>Join our newsletter
      </button>
    </section>
  </main>
  <script src="https://unpkg.com/@popperjs/core@2"></script>
  <script src="https://unpkg.com/tippy.js@6"></script>
  <script>
    document
      .querySelectorAll("[data-tooltip]")
      .forEach((element) =>
        tippy(element, { content: element.dataset.tooltip })
      );
  </script>
</body>
</html>
```

Questions