WEBDEV Proposed Path

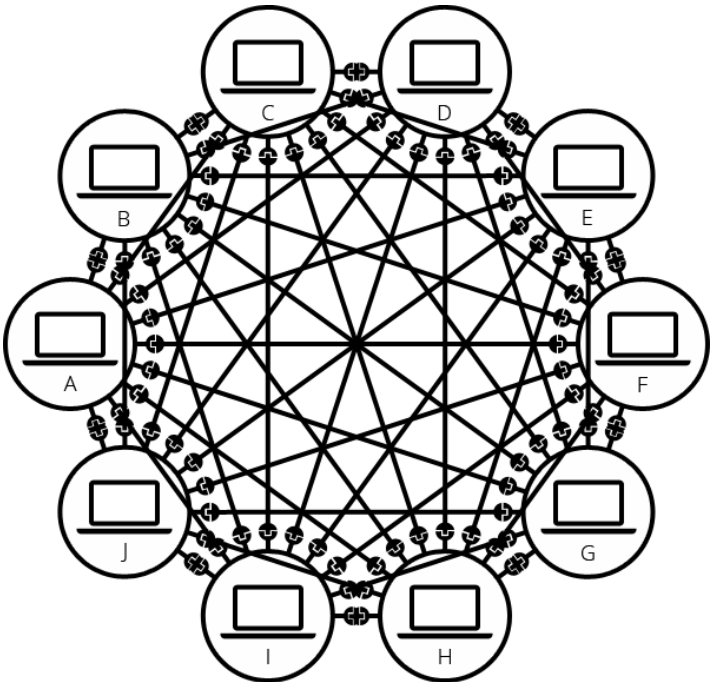
HTML
CSS
JAVASCRIPT
REACT (Vue is just easy | Angular is heavy enterprise level)
Next.js
NextAuth.js & Clerk
MongoDB
PostgreSQL (+ SupaBase +Firebase)
Node.js (+ Express + Nest.js + PrismaORM+ REST API + GraphQl)
Python Backend - Django (Enterprise) / Flask (Lightweight) / FastAPI (Rising)
Go Backend for Fast Parralel Processing
Web Sockets, Caching, Rate Limiting APIs
WebPage Security Protection
DevOps & CI/CD Pipelines
Containers (Docker & Kubernetes)
Jira Essentials With Agile Mindset
Cloud (AWS Cloud Practioner Essentials + AWS Solutions Architect + AWS SA PRO)
Blockchain / Web3 Complete Theoretical Understanding w/ Solidity or Rust Backend
Hackathons
Open Source Contributions

way more detailed & completely encompassing roadmaps

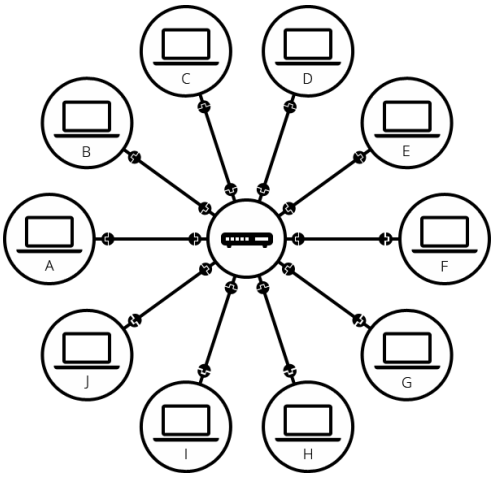
https://roadmap.sh/frontend
https://roadmap.sh/postgresql-dba
https://roadmap.sh/mongodb
https://roadmap.sh/backend
https://roadmap.sh/devops
https://roadmap.sh/docker
https://roadmap.sh/kubernetes
https://roadmap.sh/full-stack
https://roadmap.sh/blockchain
https://roadmap.sh/ux-design

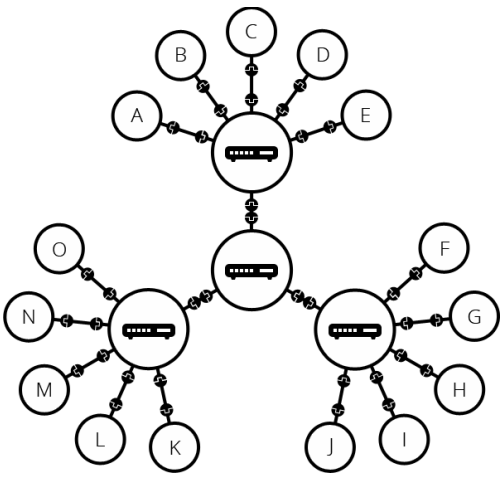
internet

In 1830s, Samuel Morse invented the Telegraph and Morse code that transmitted short pieces of information via electrical signals with each letter/alphabet having its own formula. By 1960s, J.C.R Licklider proposed the idea of a universal network that would connect large computers called mainframes to share data.This was to be done by converting**(modulating)** the data into single line of 0s and 1s that can be **demodulated** at target computer while using **packet switching** to send the signal packets via multiple paths to ensure that the end user receives some information(even if its just the fact that data has been sent) in case of a
faulty path in between.



not practical for each computer to have singular connection





We use a router connect multiple computers as having all computers connect to each other separately would be very extraneous & then those routers are connected to each other via more routers. The routers now communicate to other computer networks via cables that

connect your house to the rest of the world on the **telephone infrastructure** by using the **MODEM**. This connection is made to your **INTERNET SERVICE PROVIDER(ISP)** which are set of special routers that connect to each other and other ISPs

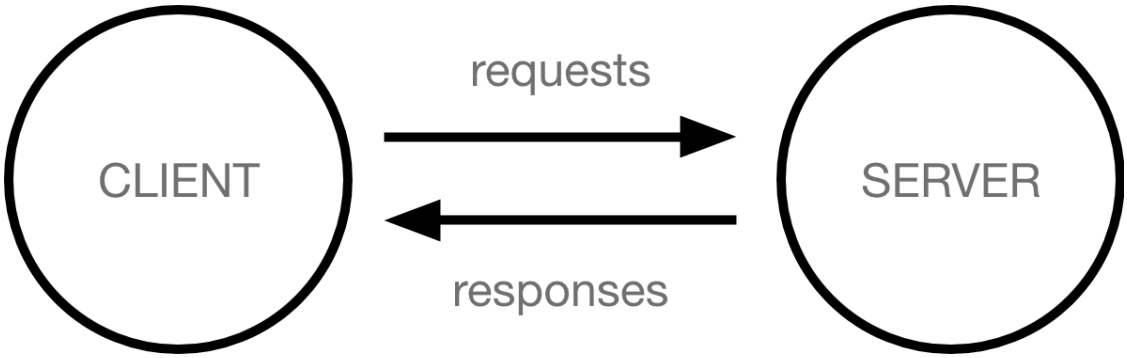
It started with **ARPANET** and then more networks of computers like France's **CYCLADES NETWORK** and London's **NPL NETWORK** emerged with each having their own set of rules on how these computers communicate called **PROTOCOL**.

In 1983, ARPANET would adopt the **Transfer Control Protocol/ Internet Protocol (TCP/IP)** giving each computer its distinct **IP Address** and the ability to automatically send packets that are lost in faulty transition paths along with the **User Datagram Protocol (UDP)** for sending data over a network where speed and efficiency is more important than sending all of the information(example - video calls having buffering or robotic voice due to packet loss) to form the **INTER NETWORK** or **INTERNET.**

Computers on the Internet are either **Clients** or **Severs.**

Clients are typical web user's internet connected devices like phone or pc

Servers are computers that store webpages, sites and addresses that are transferred to download as a copy on client's device when it requests to display it.



A **DomainNameSystem SERVER** was established as a phone book(network of computers that store the word to address conversions) as simple text for replacement of complex IP Addresses in format of ServerName.LocationofServer (example .gov for government and .com for commercial with each word corresponding to a number that is the real adress)

WorldWideWeb(W3/WWW) is a server with more technologies like **Hyper Text Protocol Language(http)** & **Hypertext Markup Language(html)** which enabled data to be shared as webpages and hyperlinks that can be accessed by **Web Browsers** (Chrome , Firefox) that are SoftWare Applications that convert the words to machine readable requests and displays the result in HTML while **Search Engines** (Google, Yahoo) uses algorithms to display the most relevant results from given query.

HTML files requested by browsers often have <link> for external CSS stylesheets and <script> for referencing external JavaScript scripts. HTML is raw information like text, links, lists and buttons that website built on whereas CSS positions that information, gives it color

and changes its font and improves the look. Java script adds interactive features and complexity to the webpage with actual programs unlike HTML & CSS

command line

$ / **%** is indication to write the commands after the symbol

tab is used for auto completion of directories and files, returns options or final file left

pwd gives present working directory or where you are navigating currently

cd directoryname moves to the directory inputted

cd ~ moves to the home directory

cd – goes back to previous directory

ls gives all files inside the pwd

ls /Us[tab]/ilvo[tab] to list files without leaving current directory

ls -lh details in human readable data format like 4.5k bytes instead of 4500 in **ls -l**

ls -lhS sorts all files

ls -lr reverses sort form

ls -lt last modified order

mkdir creates a directory to stores files (same as a folder)

cp filename.format directory copies filename to directory

rm deletes files (returns error for rm directory as it cant perform the action)

rm -r deletes directories including the subdirectories inside it

mv filename.format directory cuts the file to directory

mv filename.format filename2 renames filename to filename2

touch filename.format creates a file in cd of format mentioned

open . opens the pwd in GUI

open “filename.format” opens the filename in GUI

nano filename.txt opens or creates in case it doesn’t exist a new txt file in CLI text editor

control + O and then **enter** saves files written in nano

control + X to exit nano text editor

command + t opens another tab for terminal usage

html (in the era of ai; you ask questions, don't gobble syntax)

element are pieces of content wrapped in opening and closing **tags**
some text content

void element do not have a closing tag and don't wrap around content like <img>, we can also use <img/> but its mostly historic and discouraged.

always name your homepage's html file as **index.html** as that's searched by webservers

html boilerplate

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

<!DOCTYPE html> declares version of html to render, HTML5 in this case

<html> </html> is the root element that includes all other elements and **lang** tells the language used in the element and helps improves accessibility of webpage

<head> is used for information so our website render correctly

<meta charset="UTF-8"> sets the encoding to display special characters and characters from other languages accurately

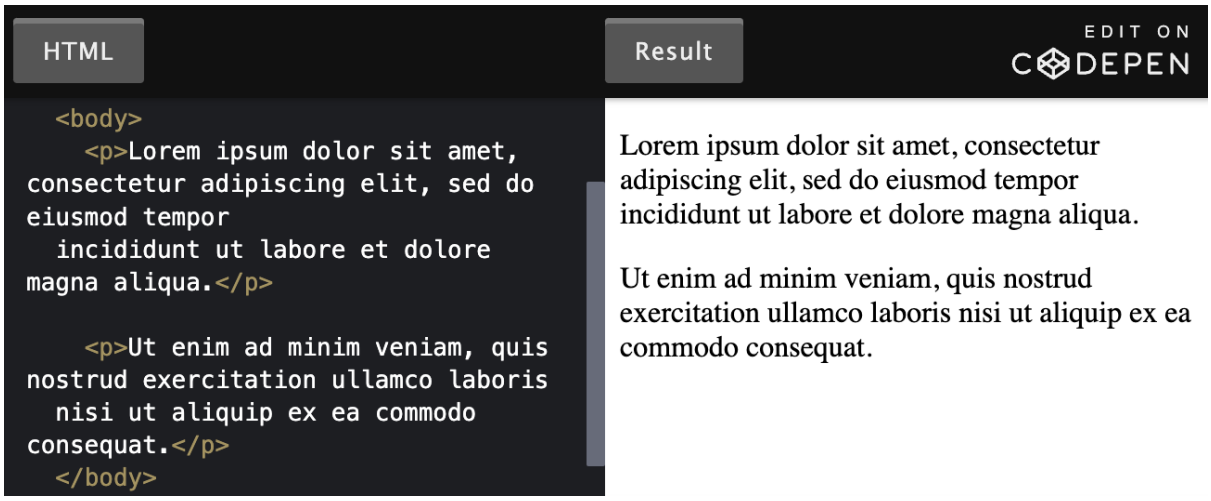
<meta name="viewport" content="width=device-width, initial-scale=1.0"> sets the width of viewport to any screen size viewing it with a scale of 1 to 1 with no zooming

<title> is used to give webpages a human readable title to display instead of index.html which means nothing to the user

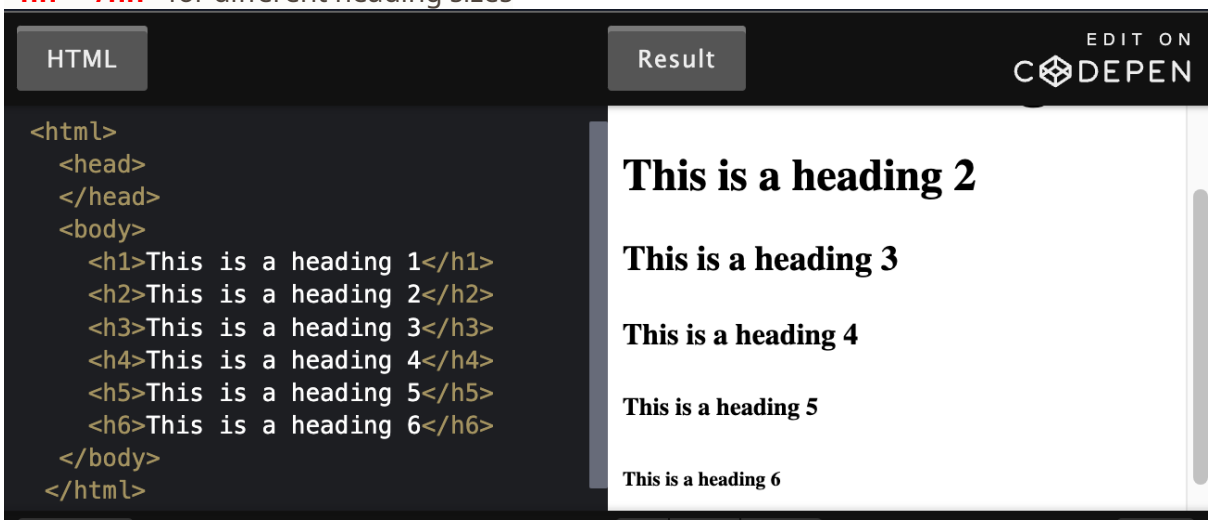
<body> is where all content to be displayed is like text, images, link and lists etc

! + enter in first line of visual studio code of html file will autocomplete to basic html boilerplate

new lines inside <body> are condensed into one whitespace hence we need to use ** ** to display paragraphs with new lines.



<hn> </hn> for different heading sizes



** ** makes the text bold in or <body> and marks it important which screen readers (tool for visually impaired users) will note

**** makes text italic

HTML elements can have **child elements** and **sibling elements** like two elements in a <body> element, we should **indent** to make the relationships clear

<!-- and **-->** syntax for html comments

```
<h1> View the html to see the hidden comments </h1>

<!-- I am an html comment -->
```

cmd + / comments/uncomments any line in visual studio code

unordered lists

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

- Item 1
- Item 2
- Item 3

ordered lists

```
<ol>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ol>
```

1. Item 1
2. Item 2
3. Item 3

links

the browser will give the element with href attribute, blue text to indicate it's a link
About The Odin Project

clicking the link will open it by default in **_self** or current tab, to open in new tab we can use About The Odin Project where the rel attribute gives the relation between current webpage and linked document (html or otherwise), here the **noopener** prevents opened link from accessing the website it was opened from and **noreferrer** prevents it from knowing which webpage or resource has a link to it. **noreferrer** encompasses **noopener** and hence can be used by itself. they are used for security reasons against phishing attacks like **tabnapping** where the new link can change the original webpage to trick users.

absolute links

they are links to other websites on internet and usually have fixed anatomy of scheme://domain/path like https://www.theodinproject.com/about where

http is scheme, the protocol used to share information, it can be **https:** or **http:** (unsecure version) but we can also have other protocols like mail clients **mailto:**

domain name indicates the requested web server and can also be written as ip address it can also have a number **www.example.com:(80)** to indicate the gate used access but its usually defaulted to 80 for http and 443 for https.

the path is path to the resource on webserver and in the past was the actual location on computer hosting information but today is just a abstraction without physical reality.

relative links

since index.html is usually in root directory, we can directly write the path to the linked document from the root directory to link it without domain or scheme.

```
<a href="./pages/about.html">About</a>
<a href="pages/about.html">About</a>
```

both should work, but **./** is preferred and the path is taken from the folder, the webpage is located in. if the current webpage is in some child folder but the href needs to be taken to a different branch of the parent directory, we can use **../**

```
../images/dog.jpg
```

images

we use a void element with no closing tag for images.

<img src = “location” alt = “information incase img doesn’t load” height = “310” width = “310”>

src is source and can be relative or absolute path

<img src="../images/dog.jpg">

https://www.theodinproject.com/mstile-310x310.png

alt is alternative for screenreading disabled users or when the image doesn’t load

height and width are also good practice even if we will resize using CSS

SVG

https://www.theodinproject.com/lessons/node-path-intermediate-html-and-css-svg

It’s a generic container usually used to contain other elements that are grouped together, It provides a method to add changes to all of the elements inside a specific container. container also have various classes which are basically naming attributes that makes it easier to apply changes like CSS to that specific set of elements.

```
<body>

    <div class="container">
        <h2>Title</h2>
        <p>This is a paragraph inside a div.</p>
    </div>

</body>
```

forms

The form element is a container element like div that wraps al the input a user will interact with on a form, it accepts two essential attributes; the first is the action attribute which takes a URL value that tells the form where it should send its data to be processed. The second is method attribute which tells the browser which HTTP request method (https://developer.mozilla.org/enUS/docs/Web/HTTP/Reference/Methods) it should use to submit the form. The GET and POST request methods are the two you will find yourself using the most. We use GET when we want to retrieve something from a server. For example, Google uses a GET request when you search as it *gets* the search results. POST is used when we want to change something on the server, for example, when a user makes an account or makes a payment on a website.

```
<form action="example.com/path" method="post">

</form>
```

form control elements

To start collecting user data, we need to use form control elements. These are all the elements users will interact with on the form, such as text boxes, dropdowns, checkboxes and buttons. **input element** tells the browser what kind of data it should expect & how it should render the element. **label element** tells the user what kind of data they are supposed to provide & **id element** needed for **label for** to match the label to input element and must be same as the label for element. **placeholder element** demonstrates how the text should be entered in what format. **name element** acts like a variable to information you send in the backend.

```
<form action="example.com/path" method="post">
  <label for="first_name">First Name:</label>
  <input type="text" id="first_name">
</form>
```

```
<label for="first_name">First Name:</label>
<input type="text" id="first_name" placeholder="Bob...">
```

```
<label for="first_name">First Name:</label>
<input type="text" id="first_name" name="first_name">
```

you can accept input as emails, passwords, numbers, date, textareas that accept text along multiple rows and columns which can be set by programmer

https://www.theodinproject.com/lessons/node-path-intermediate-html-and-css-form-basics

dropdown menu

we can also create a dropdown list where users can select an option and send to the backend server using **select element** & **option element. selected element** is used for default option set & **optgroup labels** are used for bundling options into groups.

```
<select name="Car">
  <option value="mercedes">Mercedes</option>
  <option value="tesla">Tesla</option>
  <option value="volvo" selected>Volvo</option>
  <option value="bmw">BMW</option>
  <option value="mini">Mini</option>
  <option value="ford">Ford</option>
</select>
```

```
<select name="fashion">
  <optgroup label="Clothing">
    <option value="t_shirt">T-Shirts</option>
    <option value="sweater">Sweaters</option>
    <option value="coats">Coats</option>
  </optgroup>
  <optgroup label="Foot Wear">
    <option value="sneakers">Sneakers</option>
    <option value="boots">Boots</option>
    <option value="sandals">Sandals</option>
  </optgroup>
</select>
```

select dropdowns are good for extensive list of options but for fewer options we can display them on page as **radio buttons**. they help us create multiple options that users can choose one of. When we select one radio button, it deselects any other radio button with same name attribute. **checked attribute** helps setting the default radio button set.

```
<h1>Ticket Type</h1>
<div>
  <input type="radio" id="child" name="ticket_type" value="child">
  <label for="child">Child</label>
</div>

<div>
  <input type="radio" id="adult" name="ticket_type" value="adult">
  <label for="adult">Adult</label>
</div>

<div>
  <input type="radio" id="senior" name="ticket_type" value="senior">
  <label for="senior">Senior</label>
</div>
```

checkboxes are similar to radio buttons (input type is checkbox) but they allow multiple options selected at same time. we can also have a single checkbox for true/false questions. the checked attribute also works here as explained before.

```
<h1>Pizza Toppings</h1>

<div>
  <input type="checkbox" id="sausage" name="topping" value="sausage">
  <label for="sausage">Sausage</label>
</div>

<div>
  <input type="checkbox" id="onions" name="topping" value="onions">
  <label for="onions">Onions</label>
</div>

<div>
  <input type="checkbox" id="pepperoni" name="topping" value="pepperoni">
  <label for="pepperoni">Pepperoni</label>
</div>

<div>
  <input type="checkbox" id="mushrooms" name="topping" value="mushrooms">
  <label for="mushrooms">Mushrooms</label>
</div>
```

button element

creates clickable buttons, there can be three types of button types; **submit buttons** submit the form they are contained it, upon pressing. The button element is submit button by default.

```
<button type="submit">Submit</button>
```

Reset buttons resets all data submitted by user in the form and reverts back to default initial state.

```
<button type="reset">Reset</button>
```

Generic buttons can be used for absolutely anything but mostly used in JS to create interactive UI elements.

```
<button type="button">Click to Toggle</button>
```

fieldsets & legends

it can get overwhelming & discouraging to deal with many form elements, so html provides us with capabilities to group forms elements into logical units called **fieldsets** & these fieldsets can be given a title of sorts for ease of use called **legend**.

```
<fieldset>
  <legend>Contact Details</legend>

  <label for="name">Name:</label>
  <input type="text" id="name" name="name">

  <label for="phone_number">Phone Number:</label>
  <input type="tel" id="phone_number" name="phone_number">

  <label for="email">Email:</label>
  <input type="email" id="email" name="email">
</fieldset>

<fieldset>
  <legend>Delivery Details</legend>

  <label for="street_address">Street Address:</label>
  <input type="text" id="street_address" name="street_address">

  <label for="city">City:</label>
  <input type="text" id="city" name="city">

  <label for="zip_code">Zip Code:</label>
  <input type="text" id="zip_code" name="zip_code">
</fieldset>
```

applying CSS on forms

Each browser has its own default styles for form controls, making your forms visually different for users depending on what browser they are using. To have a consistent design among all browsers, we have to override these default styles and style them ourselves.

Text-based form controls like text, email, password and text areas are reasonably straightforward to style. They operate like any other HTML element, and most CSS properties can be used on them.

Things get more tricky when creating custom styles for radio buttons and checkboxes. But there are many guides out there you can use to achieve your desired design, such as this guide on custom checkbox styling. There have also been new CSS properties made available in recent times to make styling radio buttons and checkboxes much easier. Certain aspects of other elements are downright impossible to style, for example, calendar or date pickers. If we want custom styles for these, we will have to build custom form controls with JavaScript or use one of the many JavaScript libraries that provide us with ready-made solutions.

https://internetingishard.netlify.app/html-and-css/forms/index.html

https://developer.mozilla.org/enUS/docs/Learn_web_development/Extensions/Forms#form_styling_tutorials

form validation

Validations allow us to set specific constraints or rules that determine what data users can enter into an input. When a user enters data that breaks the rules, a message will appear, providing feedback on what was wrong with the entered data and how to fix it.

Validations are a vital ingredient in well-designed forms. They help protect our backend systems from receiving incorrect data, and they help make the experience of interacting with our form as dead-stupid-simple as possible for our users.

required attribute

required attribute gives us specific fields that need to filled mandatorily before submission of form

```
<form action="#" method="get">
  <div>
    <label for="user_email">*Email:</label>
    <input type="email" id="user_email" name="user_email" required>
  </div>
  <br>

  <div>
    <label for="user_password">*Password:</label>
    <input type="password" id="user_password" name="user_password" required>
  </div>

  <button type="submit">Login</button>
</form>
```

we should also add a asterix before the label elements of these recquired attributes for ease of user experience.

we can also add text length validations, number range validations, ensure the data inputted by user follows a set pattern validation & have different styles for valid & invalid inputs like green and red for indicating the validity of data using pseudo classes.

https://www.theodinproject.com/lessons/node-path-intermediate-html-and-css-form-validation

https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Forms/Form_validation

https://www.sitepoint.com/html-forms-constraint-validation-complete-guide/

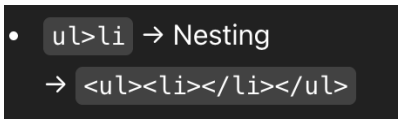
https://www.silocreativo.com/en/css-rescue-improving-ux-forms/

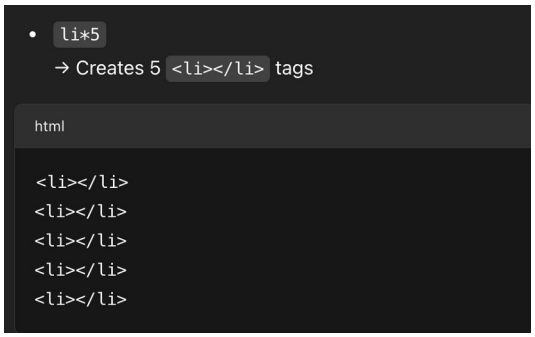
tables

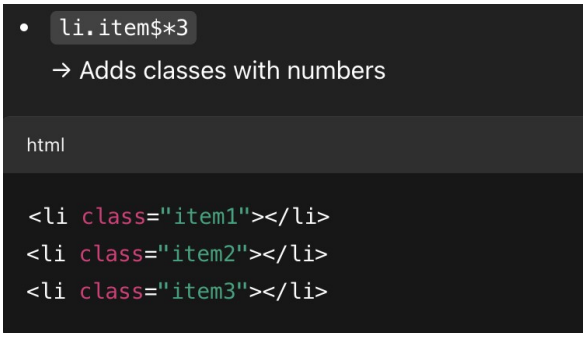
https://css-tricks.com/complete-guide-table-element/

emmet

Cmd + M (User Set) **Cmd + Shift + P** into **Search Emmet Wrap** (Wrapping Abbreviation)







Cmd + H (User set) **Cmd + Shift + P** into **Search Emmet Rem__** (Remove Tag where cursor)

https://docs.emmet.io/cheat-sheet/
https://www.youtube.com/watch?v=V8vizNQKtx0

CSS

At the most basic level, they are made of **selectors and declarations separated by semi colons**. Selector selects the elements that will be affected by CSS and declarations are the properties of which, we will change.

```
<!-- index.html -->

<div>Hello, World!</div>
<div>Hello again!</div>
<p>Hi...</p>
<div>Okay, bye.</div>
```

```
/* styles.css */

div {
  color: white;
}
```

here all three elements will be displayed in white

classes

```
<!-- index.html -->

<div class="alert-text">Please agree to our terms of service.</div>
```

```
/* styles.css */

.alert-text {
  color: red;
}
```

only the class will be affected
class names must be without whitespace, we can assign many classes to a div container separated by whitespace ** ** .box{} .rounded{} .highlight{}

```
* {
  color: purple;
}
```

we can also have a universal selector that selects elements of any type

ID selector

similar usage to classes, except one element can only have one ID &
ID can be attributed only once on one block of code as every element should have unique id

```
<!-- index.html -->

<div id="title">My Awesome 90's Page</div>
```

```
/* styles.css */

#title {
  background-color: red;
}
```

you should use IDs sparingly (if at all).

group selecting many classes for common styling

```
.read,
.unread {
  color: white;
  background-color: black;
}

.read {
  /* several unique declarations */
}

.unread {
  /* several unique declarations */
}
```

special selecting classes for unique styling

if we had two elements of class subsection header & subsection preview respectively and we wanted to apply CSS only to an element with both subsection and header and avoid all other subsections or headers. This will work for ID and class as well.

```
.subsection.header {
  color: red;
}
```

```
.subsection#preview {
  color: blue;
}
```

descendant combinator

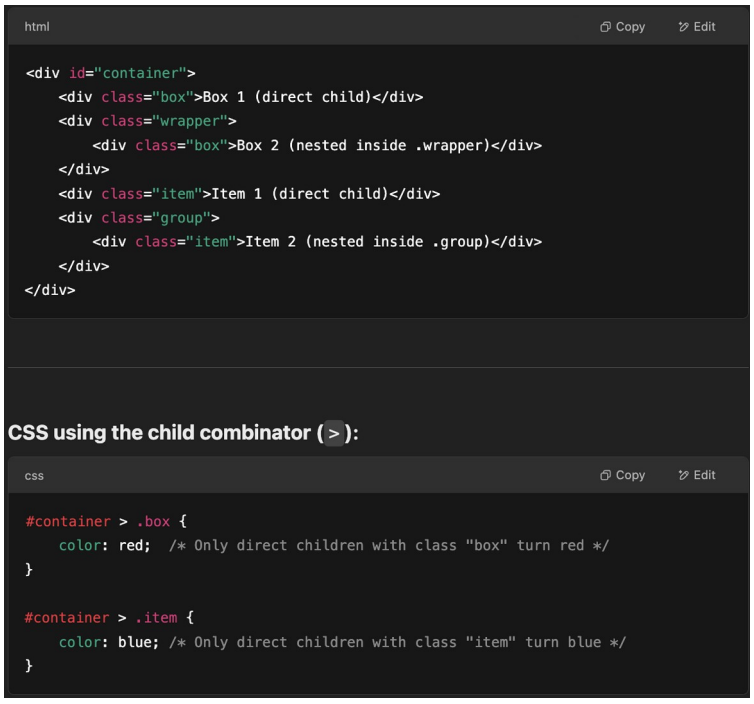
if we wanted to select a specific class that is only there inside another specific class while ignoring all the other instances of that class, we use space between the selectors in CSS. all the occurences inside parent are selected, **not just direct children.**

```html
<!-- index.html -->

<div class="ancestor">
  <!-- A -->
  <div class="contents">
    <!-- B -->
    <div class="contents"><!-- C --></div>
  </div>
</div>

<div class="contents"><!-- D --></div>
```

```css
/* styles.css */

.ancestor .contents {
  /* some declarations */
}
```

B and C are selected while D isn't

child combinator



child combinator ensures that **only direct children** are selected for CSS

some more advanced selectors

+ - adjacent sibling combinator

~ - general sibling combinator

```
<main class="parent">
  <div class="child group1">
    <div class="grand-child group1"></div>
  </div>
  <div class="child group2">
    <div class="grand-child group2"></div>
  </div>
  <div class="child group3">
    <div class="grand-child group3"></div>
  </div>
</main>
```

main div{} will select all descendants of main that are div

main>div{} will select only the immediate childs of main that are div

main>div>div{} will select the div grandchildren of main from div parents

group1+div{} will select immediate div sibling of group1 -> child group2

group1+div+div{} will select second immediate div sibling of group1 -> child group2

group1~div{} will select all group1 siblings that are also div

pseudo class

pseudo class is a selector where we are selecting elements in a specific state like if the element is being hovered over or clicked or first child same in given element type.

HTML

```
<article>
  <p>
    Veggies es bonus vobis, proinde vos postulo essum magis kohlrabi welsh onion
    daikon amaranth tatsoi tomatillo melon azuki bean garlic.
  </p>

  <p>
    Gumbo beet greens corn soko endive gumbo gourd. Parsley shallot courgette
    tatsoi pea sprouts fava bean collard greens dandelion okra wakame tomato.
    Dandelion cucumber earthnut pea peanut soko zucchini.
  </p>
</article>
```

CSS

```
article p:first-child {
  font-size: 120%;
  font-weight: bold;
}
```

we have **:last-child** & **:only-child** for elements with no siblings

:nth-child() for flexible selection

```
.myList:nth-child(5) {/* Selects the 5th element with class myList */}

.myList:nth-child(3n) { /* Selects every 3rd element with class myList */}

.myList:nth-child(3n + 3) { /* Selects every 3rd element with class myList, beginning with the 3rd */}

.myList:nth-child(even) {/* Selects every even element with class myList */}
```

- `:hover` — mentioned above; this only applies if the user moves their pointer over an element, typically a link.
- `:focus` — only applies if the user focuses the element by clicking or using keyboard controls.

`:active` applies to elements that are currently being clicked, and is especially useful for giving your user feedback that their action had an effect. This is a great one to give your buttons and other interactive elements more 'tactile' feedback.

:visited for links that have been already visited or clicked by

```
a:visited {
  color: purple;
}
```

:root represents the root element at the very top like html but unlike html, it has higher precedence due to being a pseudo class

```css
:root {
  color: red
}
html {
  color: green;
}
```

```html
<div>hello world</div>
```

hello world

pseudo elements

pseudo elements act like pseudo class **except they act like they are their own html element**; they start with double colon

HTML Play

```html
<article>
  <p>
    Veggies es bonus vobis, proinde vos postulo essum magis kohlrabi welsh onion
    daikon amaranth tatsoi tomatillo melon azuki bean garlic.
  </p>

  <p>
    Gumbo beet greens corn soko endive gumbo gourd. Parsley shallot courgette
    tatsoi pea sprouts fava bean collard greens dandelion okra wakame tomato.
    Dandelion cucumber earthnut pea peanut soko zucchini.
  </p>
</article>
```

CSS Play

```css
article p::first-line {
  font-size: 120%;
  font-weight: bold;
}
```

::marker allows you to customize the styling of bullets/numbers in <li> elements

::first-letter / **::first-line** allows you to give special styling to first letter / line

::selection allows you to change the highlighting when a user selects text on page

::before / **::after** allows us to add extra elements from css before or after a html element

```
<style>
  .emojify::before {
    content: '😎 😄 🤓';
}

  .emojify::after {
    content: '🤓 😄 😎';
}
</style>

<body>
  <div> Let's <span class="emojify">emojify</span>this span!</div>
</body>
```

Let's 😎 😄 🤓 emojify 🤓 😄 😎 this span!

https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Pseudo_classes_and_elements

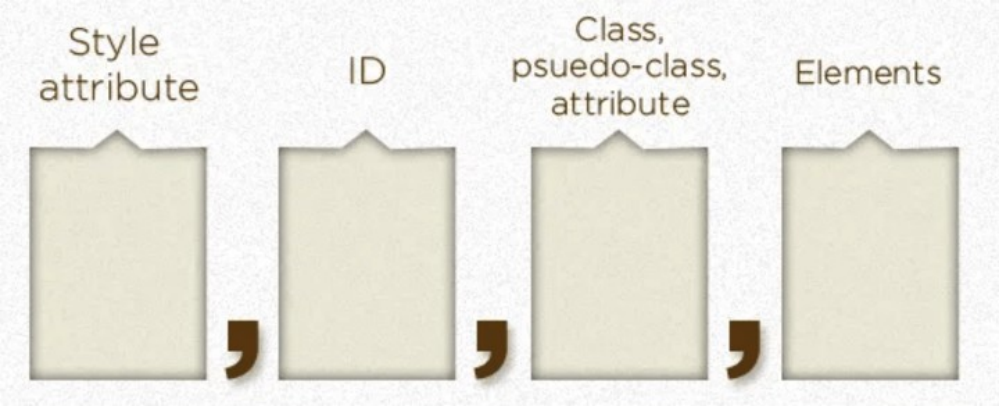
attribute selector

attributes are anything in opening tags, we can use selectors to select elements with specific opening tag, link and specific element type.

```
[src] {
  /* This will target any element that has a src attribute. */
}

img[src] {
  /* This will only target img elements that have a src attribute. */
}

img[src="puppy.jpg"] {
  /* This will target img elements with a src attribute that is exactly "puppy.jpg" */
}
```

```
[class^='aus'] {
  /* Classes are attributes too!
    This will target any class that begins with 'aus':
    class='austria'
    class='australia'
  */
}

[src$='.jpg'] {
  /* This will target any src attribute that ends in '.jpg':
  src='puppy.jpg'
  src='kitten.jpg'
  */
}

[for*='ill'] {
  /* This will target any for attribute that has 'ill' anywhere inside it:
  for="bill"
  for="jill"
  for="silly"
  for="ill"
  */
}
```

CSS specificity value



If a element has two or more types of instructions given for css property then the final displayed instruction is decided via specificity value, the values can be 0 or 1.

If the styling is inline then first number is 1 and hence it will be chosen over any styling that is not inline

If id is mentioned in style declaration then second number is 1
If class is mentioned in style declaration then third number is 1
If element type like div or span is mentioned in declaration then fourth number is 1

css declaration with larger specificity value will be used for display

https://css-tricks.com/specifics-on-css-specificity/

CSS precedence with detailed examples

by default the precedence based on specificity value is

1) ID selectors.
2) Class selectors.
3) Type selectors.

```
<!-- index.html -->

<div class="main">
  <div class="list subsection">Red text</div>
</div>
```

```
/* rule 1 */
.subsection {
  color: blue;
}

/* rule 2 */
.main .list {
  color: red;
}
```

red takes priority over Blue as it has a more specific selector than just subsection.

```
/* rule 1 */
#subsection {
  color: blue;
}

/* rule 2 */
.main .list {
  color: red;
}
```

blue will take priority as ID is more prioritised selector

```
/* rule 1 */
#subsection {
  background-color: yellow;
  color: blue;
}

/* rule 2 */
.main #subsection {
 color: red;
}
```

rule 2 takes precedence as it has more specificity , but the background color from Rule 1 will be applied as there is no conflicting declarations

```css
/* rule 1 */
.class.second-class {
  font-size: 12px;
}

/* rule 2 */
.class .second-class {
  font-size: 24px;
}
```

in this case since both rules have two classes, they are same in specificity, in such cases later/last declaration is always selected.

```css
/* rule 1 */
* {
  color: black;
}

/* rule 2 */
h1 {
  color: orange;
}
```

orange will be used as Universal Selector has no precedence while Type selector has lowest.

```html
<!-- index.html -->

<div id="parent">
  <div class="child"></div>
</div>
```

```css
/* styles.css */

#parent {
  color: red;
}

.child {
  color: blue;
}
```

despite ID having more precedence, blue will be used over red as it's a direct target

CSS properties

https://css-tricks.com/almanac/properties/

color

```
color: red;
```

```
/* hsl example: */
color: hsl(15, 82%, 56%);
```

```
/* rgb example: */
color: rgb(100, 0, 127);
```

```
/* hex example: */
color: #1100ff;
```

font-family and properties

```
font-family: "Times New Roman", Georgia, serif;
font-size: 22px;
font-weight: 700;
text-align: center;
```

quotations is used for Times New Roman because it has spaces in it.
we mention three fonts for cases where browser cannot support the first font and so on.
serif is the best fallback because at best, serif is supported on most operating systems.

font-families, using web fonts libraries, font style, line height, letter spacing & overflowing text / ellipsis

https://www.theodinproject.com/lessons/node-path-intermediate-html-and-css-more-text-styles

border https://developer.mozilla.org/en-US/docs/Web/CSS/border

border radius is the property that is used to create rounded corners on elements.

https://developer.mozilla.org/en-US/docs/Web/CSS/border-radius

background

the background property is shorthand for 8 specific properties, beyond changing background-colors, you can also specify background images, change the position and size of background images, and change how background images repeat or tile if they are too small to fill their container. It is also possible to have multiple background layers. One thing to note is that it *is* possible to use these properties individually, and in some cases it might be easier and more clear to do that than defaulting to the shorthand.

https://developer.mozilla.org/en-US/docs/Web/CSS/background **(note :formal syntax section is machine readable syntax for CSS & is very deep level CSS theory)**

boxshadow is the property that adds shadow element around a element.
https://developer.mozilla.org/en-US/docs/Web/CSS/box-shadow

text overflow https://developer.mozilla.org/en-US/docs/Web/CSS/overflow

opacity https://developer.mozilla.org/en-US/docs/Web/CSS/opacity

image properties

by default they will take the original image height & width but we can change their values

```
<body>

    <h2>Fixed Width and Height</h2>
    <img src="https://via.placeholder.com/300x200" class="fixed-size" alt="Fixed Size Image">

    <h2>Fixed Width, Auto Height</h2>
    <img src="https://via.placeholder.com/300x200" class="auto-height" alt="Auto Height Image">

    <h2>Percentage Width</h2>
    <img src="https://via.placeholder.com/300x200" class="percentage-size" alt="Percentage Width Image">

</body>
```

```
<style>
    .fixed-size {
        width: 300px;
        height: 200px;
    }

    .auto-height {
        width: 300px;
        height: auto;
    }

    .percentage-size {
        width: 50%;
        height: auto;
    }
</style>
```

auto sets the value based on the aspect ratio, we should always mention the properties coz if the image took longer to load, it will cause a sudden drastic shift in how the webpage was looking causing bad user experience.

CSS Size

size is represented in length units (absolute or relative)

absolute units

are always same in any context, **px** is an absolute unit because the size of a pixel never changes, it is the only absolute unit you should be using.

relative units

em is relative to font size of an element so if width or any property in css is given in em value, it will be relative to font size times the number written before em, if the font size property is given in em value then its relative to font size of parent element

rem is relative to font size of root element

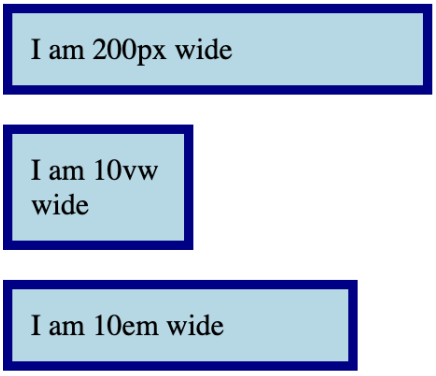
vw vh are relative to viewport width and height.

HTML

```html
<div class="wrapper">
  <div class="box px">I am 200px wide</div>
  <div class="box vw">I am 10vw wide</div>
  <div class="box em">I am 10em wide</div>
</div>
```

CSS

```css
.box {
  background-color: lightblue;
  border: 5px solid darkblue;
  padding: 10px;
  margin: 1em 0;
}

.wrapper {
  font-size: 1em;
}

.px {
  width: 200px;
}

.vw {
  width: 10vw;
}

.em {
  width: 10em;
}
```



If you change the font size in .wrapper from 1em to 1.5em; then the .em element will increase in width and get wider because its relative to font size

here 1em will be font size of parent element which if not given is just **16px**

10 vw is 10% of viewport width

```
<ul class="ems">
  <li>One</li>
  <li>Two</li>
  <li>
    Three
    <ul>
      <li>Three A</li>
      <li>
        Three B
        <ul>
          <li>Three B 2</li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

```
<ul class="rems">
  <li>One</li>
  <li>Two</li>
  <li>
    Three
    <ul>
      <li>Three A</li>
      <li>
        Three B
        <ul>
          <li>Three B 2</li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

```
html {
  font-size: 16px;
}

.ems li {
  font-size: 1.3em;
}

.rems li {
  font-size: 1.3rem;
}
```

- One
- Two
- Three
  - Three A
  - Three B
    - Three B 2
- One
- Two
- Three
  - Three A
  - Three B
    - Three B 2

since em is relative to parent elements font size; in descendant elements the size is relatively increasing whereas for rem size; the size is always relative to root parent element.

you can also set value part of **css property pair as percentages** relative to font-size of parent element.

https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Values_and_units

https://css-tricks.com/fun-viewport-units/

default style

Each browser has their own set of default styles called user agent stylesheets, written CSS always has higher precedence but if you want to reset your default CSS.
https://mattbrictson.com/blog/css-normalize-and-reset

linking CSS to HTML

i) external CSS

separate external css file linked in <head> container

```
<!-- index.html -->

<head>
  <link rel="stylesheet" href="styles.css">
</head>
```

styles.css used coz both files in same directory but we must give proper location otherwise

rel attribute is important for specification of relationship between the html and linked file

```
/* styles.css */

div {
  color: white;
  background-color: black;
}
```

ii) internal CSS

```
<head>
  <style>
    div {
      color: white;
      background-color: black;
    }

    p {
      color: red;
    }
  </style>
</head>
```

iii) inline CSS (usually for quick testing & overrides other CSS; avoid)

```
<div style="color: white; background-color: black;">...</div>
```

CSS Variables

they allow us to reference a value however many times we want throughout a file & save us from changing every instance in case of a change needed by just changing the value of the variable. variables are declared after a double hyphen & require dash for spacing in variable name. we use var() function to access the variable.

```
.error-modal {
  --color-error-text: red;
  --modal-border: 1px solid black;
  color: var(--color-error-text);
  border: var(--modal-border);
```

var() function takes in two values as arguments, the needed value and a fallback value in case the first value is not yet declared or invalid for css property, this fallback value can be another css variable with its own fallback value as well.

```
.fallback {
  --color-text: white;

  background-color: var(--undeclared-property, black);
  color: var(--undeclared-again, var(--color-text, yellow));
}
```

scope

```
<div class="cool-div">
  <p class="cool-paragraph">Check out my cool, red background!</p>
</div>

<p class="boring-paragraph">I'm not in scope so I'm not cool.</p>
```

```
.cool-div {
  --main-bg: red;
}

.cool-paragraph {
  background-color: var(--main-bg);
}

.boring-paragraph {
  background-color: var(--main-bg);
}
```

only the element with the cool-paragraph class would get styled with a red background since it's a descendant of the element where our css variable is declared. If we want to use variables in all code scope then we can declare it in :root{}

https://css-tricks.com/a-complete-guide-to-custom-properties/

CSS functions

Similar to programming languages, functions in CSS are reusable pieces of code which perform specific tasks. Functions are passed "arguments" between parentheses, each of which is used by the function in a specific way. There is no way to create our own functions and we can only use premade functions like

```
color: rgb(0, 42, 255);
background: linear-gradient(90deg, blue, red);
```

calc()

helps in assigning values directly from calculations

```
:root {
--header: 3rem;
--footer: 40px;
--main: calc(100vh - calc(var(--header) + var(--footer)));
}
```

min()

Focus on this line width: min(150px, 100%); we can make several observations: If there are 150px available to the image, it will take up all 150px. If there are not 150px available, the image will switch to 100% of the parent's width.

max()

max() works the same way as min, only in reverse. It will select the largest possible value from within the parentheses. You can think of max() as ensuring a *minimum* allowed value for a property.

```
width: max(100px, 4em, 50%);
```

clamp()

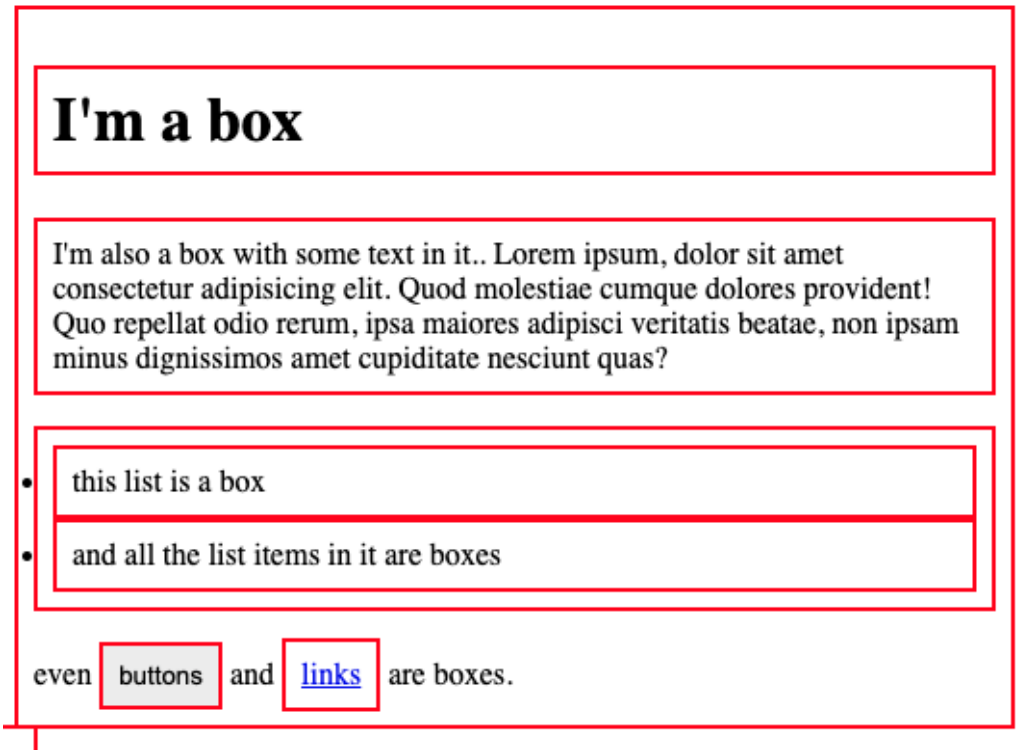
clamp() takes three values (smallest possible value, ideal value, maximum value);

```css
h1 {
  font-size: clamp(320px, 80vw, 60rem);
}
```

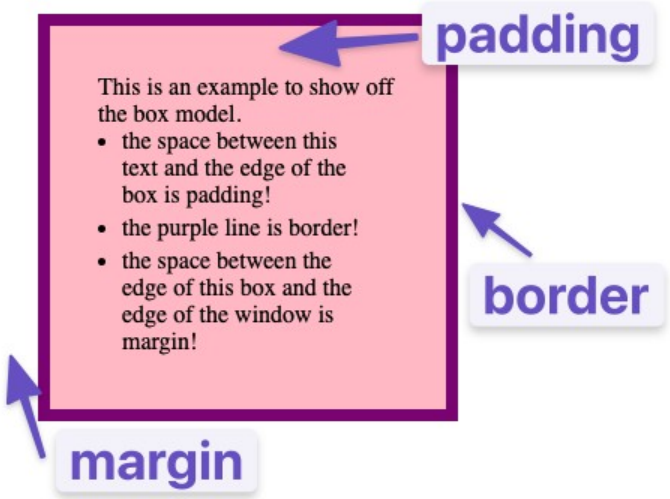
https://developer.mozilla.org/enUS/docs/Web/CSS/CSS_Values_and_Units/CSS_Value_Functions

boxing

Every element in a webpage is a rectangular box. This gets more evident when we apply universal ***{outline : 2px solid red;}**



- `padding` increases the space between the border of a box and the content of the box.
- `border` adds space (even if it's only a pixel or two) between the margin and the padding.
- `margin` increases the space between the borders of a box and the borders of adjacent boxes.



block and inline

CSS has two box types : **{display : block}** and **{display : inline;}** for example -

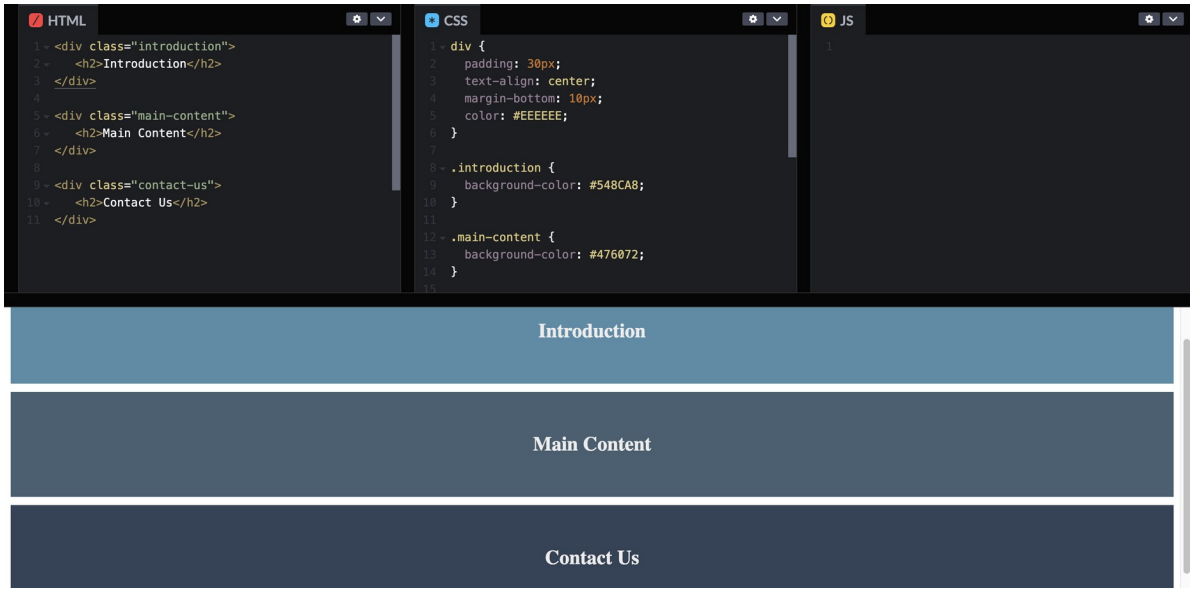
```css
p {
  display: inline;
}
```

Now, the `<p>` elements will be inline and appear next to each other.

blocks are stacked upon each other while inline will just start wherever mentioned like a link
,the padding and margin works differently for inline elements and hence aren't touched.

{display: inline-block} works like inline elements but with block style padding and margins.

** ** is a container block element by default



**** is a container inline element by default

flexbox

```html
<div class="flex-container">
  <div class="one"></div>
  <div class="two"></div>
  <div class="three"></div>
</div>
```

```css
.flex-container {
   display: flex;
}

/* this selector selects all divs
inside of .flex-container */
.flex-container div {
  background: peachpuff;
  border: 4px solid brown;
  height: 100px;
   flex: 1;
}
```

Instead of stacking the boxes over each other, by declaring display: flex; the boxes are stacked side by side with the flex-item height(.flex-container div) designating the size of flexbox.

Here flex: 1; means flex-grow: 1; flex-shrink: 1; flex-basis: 0; or flex: 1 1 0%;

flex-grow determines the width of item, so if one of the items had 2 value, then it would have double the width as other items

```css
.flex-container {
  display: flex;
}

.flex-container div {
  background: peachpuff;
  border: 4px solid brown;
  height: 100px;
  flex: 1 1 0%;
}

.flex-container .two {
  flex: 2 1 0%;
}
```

flex-shrink determines the adjust rate of items, in case the total sum of item width exceeds the width available on the display screen, higher the value, more the item will shrink in comparison.

The default shrink value is 1, so all items shrink evenly

```
.flex-container {
  display: flex;
}

.flex-container div {
  background: peachpuff;
  border: 4px solid brown;
  height: 100px;
  width: 250px;
  flex: 1 1 auto;
}
.flex-container .two {
  flex-shrink: 0;
}
```

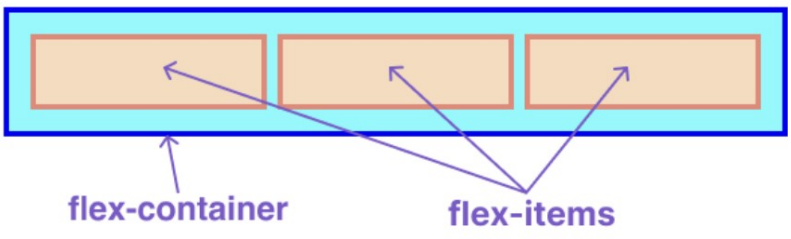
The second item will shrink the least when display size is decreased as it has a flex-shrink of 0; while others shrink a lot.

flex-basis gives the initial size of an flex item, usually defaulting to 0% meaning it starts with 0% of the size of the container and then flex grows and flex shrinks based on display width, whereas flex-basis : auto tells the item to check width declaration given in .classname div

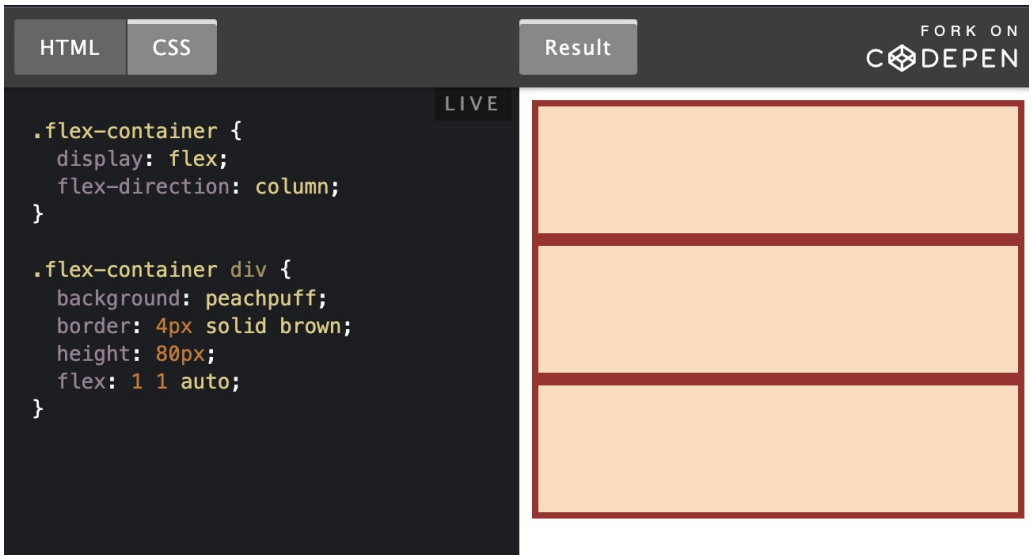
by default flex-basis is set to auto; but if we add flex:1; then it defaults to **flex: 1 1 0%;**

flex:auto; defaults to **flex: 1 1 auto;**

```
.container {
  background-color: aqua;
  padding: 12px;
  border: 4px solid blue;
  display: flex;
}

.item {
  background-color: peachpuff;
  border: 4px solid salmon;
  height: 48px;
  flex: 1;
  margin: 4px;
}
```

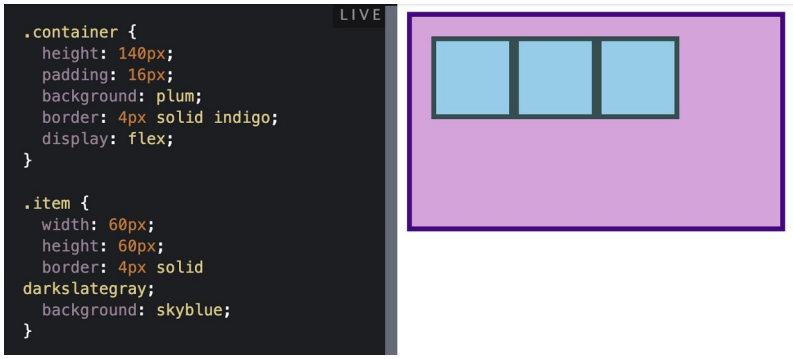


any div container with display:flex is a flex container while the .flex-container div are flex items inside the container. **flex-direction** gives the directional axes of flexbox. The default axes is horizontal or row but vertical flexbox is possible with flex-direction : column;

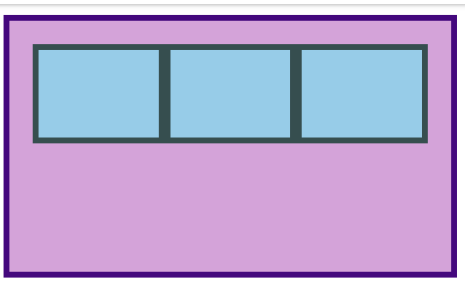


flex-basis here refers to the height and not the width which leads to **flex:1;** not working as it has **flex-basis: 0%** which gives the initial calculated value before grow and shrink to 0% of container, but these items don't need to grow at all, as the empty div containers already have 0 height so they don't have to fill up the height of their container making all items inside collapse.

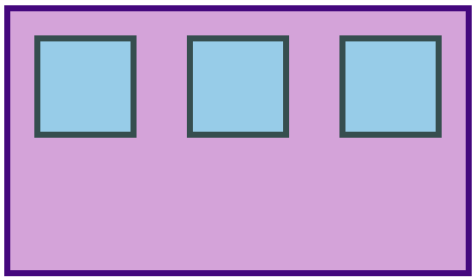
> *"There are situations where the behavior of flex-direction could change if you are using a language that is written top-to-bottom or right-to-left, but you should save worrying about that until you are ready to start making a website in Arabic or Hebrew."*



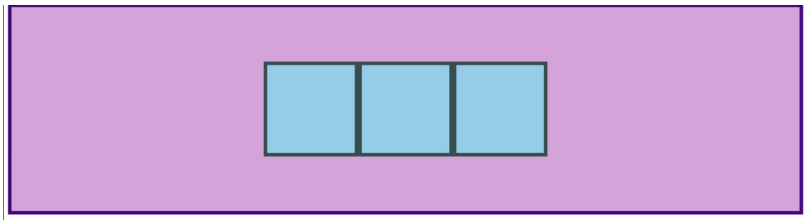
here the items are left aligned as they respect the 16px padding and without **flex:1;** they will default to height and width declarations.
adding **flex:1;**



removing flex:1; and adding **justify-content: space-between;** to the .container, which maintains height and width declarations but still spreads out along the main axis, like flex:1; justify-content controls alignment across the main axis (horizontal by default; changes for axes change) **justify-content** can also have **:centre;** and will function accordingly

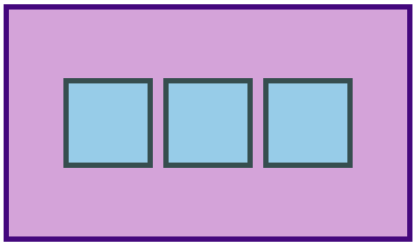


similarly to maintain the height and width declaration but align the items in the centre, we can add **align-items: center** to the .container. align-items controls alignment across the cross axis (vertical by default; changes for axes change)



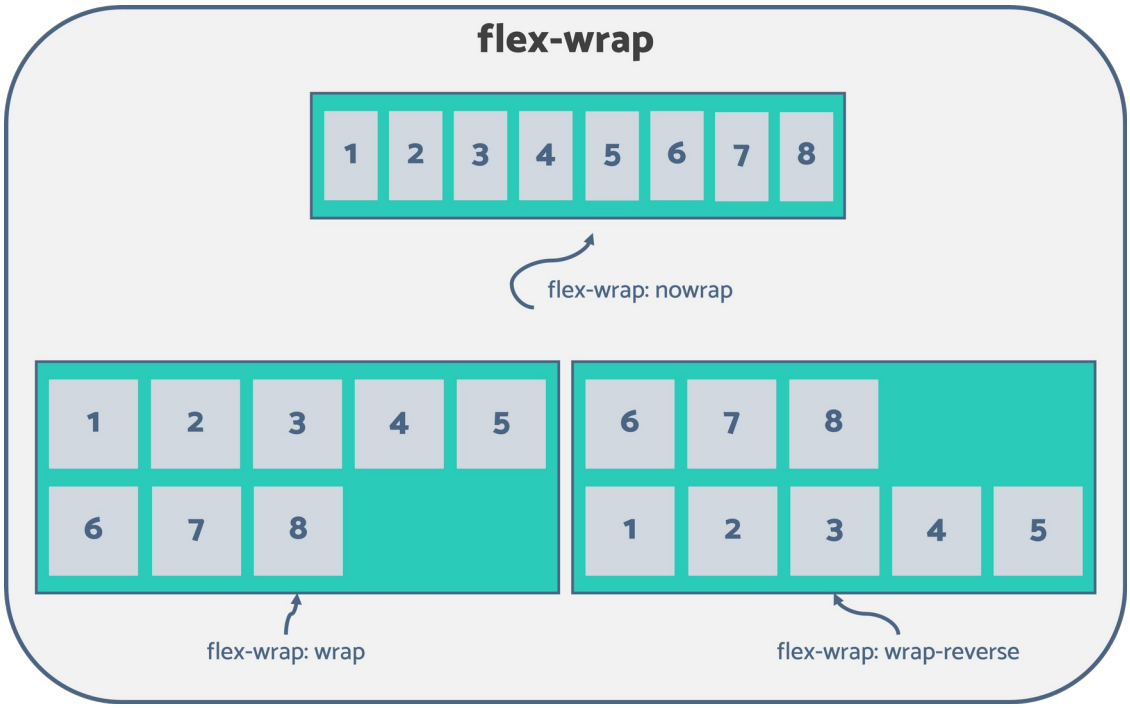
```
align-items: center;
justify-content: center;
gap: 8px;
```

we can also add **gap: 8px** to the .container which will always maintain a specified gap



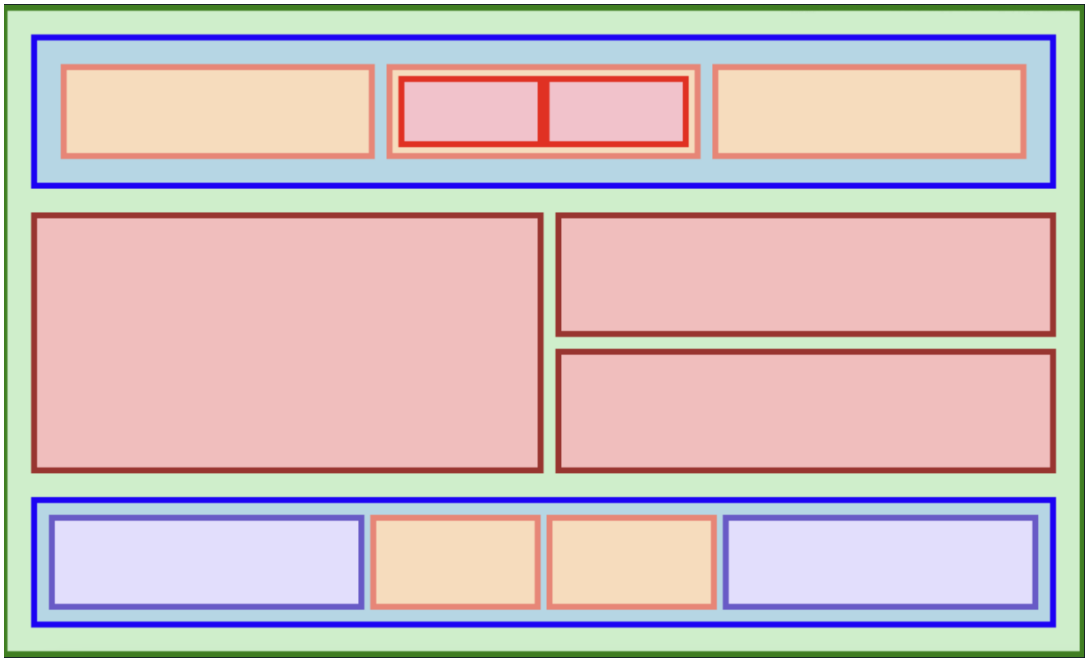
flex-wrap: allows us to use the next line for flex items, if there is not enough space in the first line. If a flex container had width 200px but 4 items with width 100px each, we can wrap them to the second line & have a square of 4 items.

this property can have three values; wrap, nowrap & wrap-reverse(items wrap above).



if the flex items have set size & cannot fit into the flex container's set size then it will overflow the container on the same line on nowrap

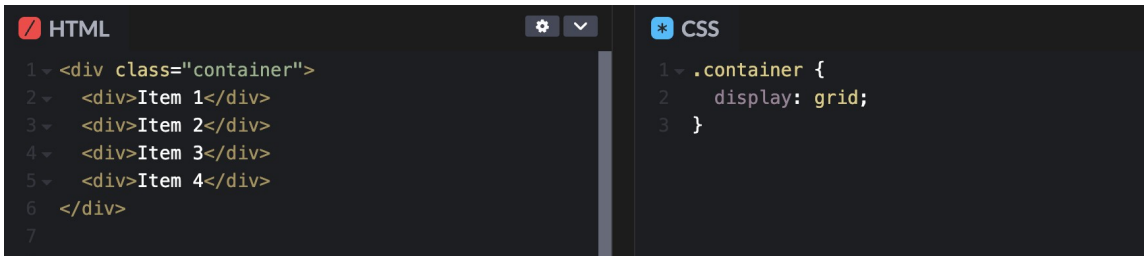
flex items can also be flex containers by applying display: flex to them and deciding the properties for their children flex items **making flexbox a very powerful tool.**



grid

grid is a layout system inspired from newspaper & magazine layouts, it proves to be another good tool besides flexbox for creating good quality website styling.

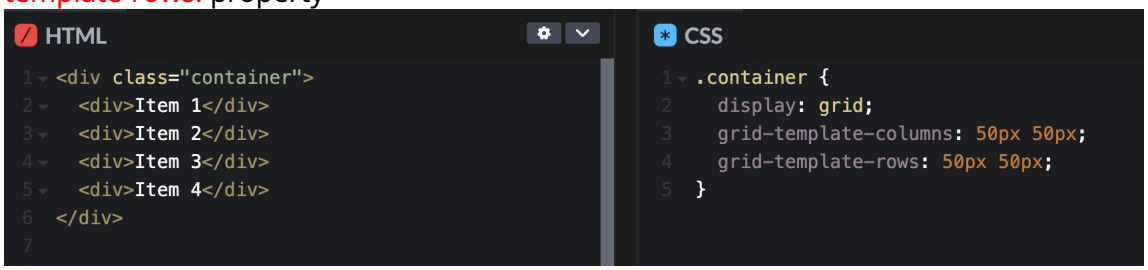
It allows us to cleanly use two dimensional placement of items (multi line),
it can start of as a one line container but gives us the ability to add multiple lines later on.
we setup a grid using display: grid; or display: inline-grid;



Item 1
Item 2
Item 3
Item 4

only the direct children are made grid items while grandchildren items are not made grid items, so if there was a grandchildren element in between, it will be placed out like a normal item & the grid cell will continue next line for remaining items. like flexbox, grid items can be made grid containers as well.

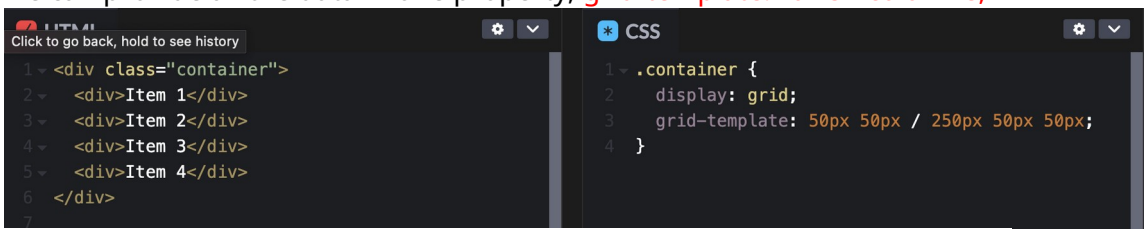
we provide number of columns/rows and their sizes in the grid-template-columns: & grid-template-rows: property



Item 1 Item 2

Item 3 Item 4

we can provide all the data in one property, grid-template: rows / columns;



Item 1 Item 2 Item 3

Item 4

the first column is really big so item 2,3 seem to be in same grid cell but that is not the case, the items are filled row wise, item 2 is in row 1 col 2 & item 3 in row 1 col 3.

due to the inline nature of grid, exceeding items will start be placed in grid layout created beyond the defined set of rows, we can set the sizes & number of these newly created grid rows using grid-auto-rows & grid-auto-columns and grid-auto-flow : column; for applying the previously used columns sizes for new rows.

grid gaps using column-gap: & row-gap:

```html
<div class="container">
  <div>Item 1</div>
  <div>Item 2</div>
  <div>Item 3</div>
  <div>Item 4</div>
</div>
```

Double-click to expand.

```css
.container {
  display: grid;
  grid-template-columns: 50px 50px;
  grid-template-rows: 50px 50px;
  column-gap: 10px;
  row-gap: 100px;
}

.container > div {
  border: 1px solid blue;
}
```

Item 1 Item 2

Item 3 Item 4

property gap: will just apply the same gap to both row-gap and column-gap.

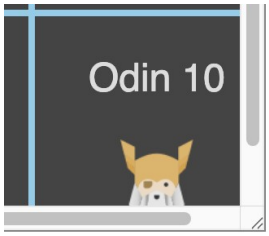
https://css-tricks.com/snippets/css/complete-guide-grid/

https://developer.chrome.com/docs/devtools/css/grid/

grid mechanics

resize: both & overflow: auto

if a container had resize:both and overflow: auto then the container can be dragged and resized from bottom right corner & if the container is dragged smaller than it can accommodate items, it will have a scroll bar to visualize all items.



repeat()

repeat() css functions reduces typing overhead of retyping all the values for many items.

```
.grid-container {
  grid-template-rows: 150px 150px;
  grid-template-columns: 150px 150px 150px 150px 150px;
}
```

can be rewritten as:

```
.grid-container {
  grid-template-rows: repeat(2, 150px);
  grid-template-columns: repeat(5, 150px);
}
```

dynamic grids

```
grid-template-rows: repeat(2, 1fr);
grid-template-columns: repeat(2, 2fr) repeat(3, 1fr);
```

here all rows will be given 1 fraction of the size of the grid container while the first columns will be given 2 fractions & last 3 given 1 fractions. If were to drag and change the size of the container, the items will increase/decrease in size maintaining this fractional ratio.


```
.grid-container {
  grid-template-rows: repeat(2, min(200px, 50%));
  grid-template-columns: repeat(5, max(120px, 15%));
}

.grid-container {
  grid-template-rows: repeat(2, 1fr);
  grid-template-columns: repeat(5, minmax(150px, 200px));
}
```

we can also set minimum values & maximum values for our grid items. (% is % of height/ width of container). minmax() function sets minimum and maximum value for grid item

autofitting & auto filling based on minimum size to grid container size

flexbox vs grid

A way to decide between Grid and Flexbox is to consider if your design starts from the content, or from the layout:

In Content First Design, you begin with clarity of how the content should be, and the layout follows. This is a great opportunity to use Flexbox. Its flexible nature gives you control of the *behaviour* of items through logical rules. How they grow, shrink, their ideal size and position in relation to each other, all make the layout dynamic. While Flexbox gives you control over its content, the final layout is only a consequence. Depending on the dimensions of the flex container, the general layout can change a lot.In Layout First Design, you decide how you want the pieces arranged, then fill in the content. That is when Grid shines. Defining grid row and column tracks gives you full control of layout. Content in a grid can only fill the spaces of explicit or implicit tracks. So, when you have an idea of how the big picture of a container should look like, Grid is the clear choice.

If you have one-dimensional content, Flexbox can make it easier to control how that content is positioned in a Flex container. If, on the other hand, you want to accurately place content on a complex layout in two-dimensions, Grid can be easier to use. Say you want your overall layout to be a grid, but you want the grid items to act as flex parents. This way, the grid items can be moved around using the precise two-dimensional placement Grid allows for, while also allowing the content inside the grid items to be capable of freely moving around using Flex.

HTML

```html
<div class="layout">
  <div class="child">
    <div></div>
    <div></div>
    <div></div>
  </div>
  <div class="child">
    <div></div>
    <div></div>
    <div></div>
  </div>
  <div class="child last-row">
    <div></div>
    <div></div>
    <div></div>
  </div>
</div>
```

CSS

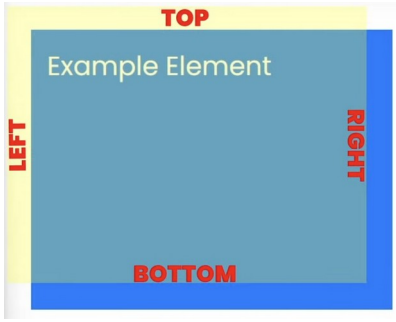
```css
.layout {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-gap: 8px;
}
.child {
  background: orange;
  text-align: center;
  padding: 25px;
  display: flex;
  justify-content: space-between;
}
.last-row {
  grid-column: 1 / 3;
}
.child > div {
  height: 50px;
  background: darkred;
  flex: 0 1 30%;
}
```

positioning

moving elements on the screen using flexbox and grid & things like margin, padding have all relied on CSS default positioning mode which is mostly used for all your needs.

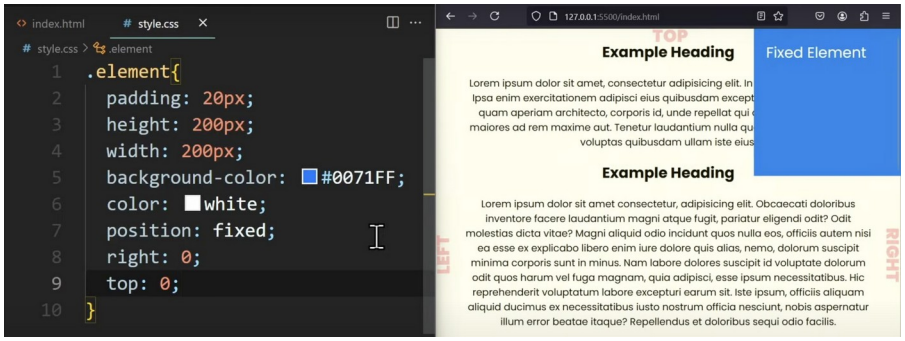
position : static; is the default positioning mode, if an element is static, it will be placed where it is placed in the order of elements: top to bottom. properties like top, right, bottom & left do not affect the position of element in any way

position: relative; is another positioning mode where properties like top, right, bottom & left will displace the item from where it would normally be placed if it were static.



top: 30px; left: 30px; will offset the element by 30px from top & left.

position: fixed; will fix the element where it exists statically, so if we scroll up or down; the element will stay where it was on the screen on a different layer of website. properties like top ,right ,bottom & left will now be relative to screen view. top: 0px will keep the element 0px; from top while right:0px will keep the element 0px from the right.



position: sticky; acts like a normal static element but while scrolling if it reaches a given value like top:0px; then it will be fixed on the screen layer ensuring it never leaves the visual layer even after scrolling past.

position: absolute; will use top,bottom,right,left parameters to position itself relative to (inside the element) closest ancestor that is not a static element. a relative element with no top,bottom,right & left parameter behaves like a static element but still provides positional context that absolutely positioned elements can use for its position. If there is no ancestor with a non-static positioning, then it will just position itself relative to page but it will be able to scrolled past from.

transforms

you can rotate, scale, skew & translate (or move relative to itself). you can also apply multiple transforms to the same items. there are some more niche transforms but they can all only be applied to elements that are replaceable (like items or inputs that are brought in from somewhere else) and are aren't inline (they don't start anywhere but are in a block form).

https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-transforms

transitions

you can change the properties of elements on specific actions like hovering over a cursor
https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-transitions

animations

- Transitions were designed to animate an element from one state to another. They *can* loop, but they weren't designed for that. Animations, on the other hand, *were* designed with the purpose of explicitly enabling loops.
- Transitions need a trigger, such as the use of pseudo-classes like :hover or :focus, or by adding/removing a class via JavaScript. Animations, on the other hand, do not need such a trigger. Once you have your elements in place and CSS defined, an animation will start running immediately if that's what you told it to do.
- Transitions are not as flexible as using animations. When you define a transition, imagine you are sending that element on a journey in a straight line from point A to point B. Yes, the transition-timing-function can add some variation to the timing of this change, but it doesn't compare to the amount of flexibility added by using animations.

https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-keyframes

web accessibility

web accessibility ensures that websites and digital content are usable by people of all abilities, including those with disabilities. It involves designing and developing content that can be navigated and understood by everyone.

1) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-introduction-to-web-accessibility

2) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-the-web-content-accessibility-guidelines-wcag

3) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-semantic-html

4) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-accessible-colors

5) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-keyboard-navigation

6) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-meaningful-text

7) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-wai-aria

8) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-accessibility-auditing

responsive design

1) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-introduction-to-responsive-design

2) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-natural-responsiveness

3) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-responsive-images

4) https://www.theodinproject.com/lessons/node-path-advanced-html-and-css-media-queries

pre processors / pre compilers

Preprocessors (aka precompilers) are languages that help you write CSS more easily. They can reduce code repetition and provide all sorts of time-saving and code-saving features, for example by allowing you to write loops and conditionals, and join multiple stylesheets.

CSS preprocessors are essentially extensions to vanilla CSS that provide some extra functionality, the code you write is converted to vanilla CSS before the compilation begins (like #define PI 3.14 in C++).

Preprocessors do have some unique and helpful tools, but many of their most helpful features have been implemented in vanilla CSS, so it might not be worth the overhead of learning one unless you think you really need these features.

CSS frameworks

the most used framework is **bootstrap CSS** but it is mostly compatible with vanilla stack of HTML CSS JS, which in modern web development context is outdated as most development is done on react or supersets of react like next.js.

tailwind CSS is rising as the most dominantly used CSS framework among new repositories, heavily preferred as top choice for front end developers and works well with react & react like js frameworks.

https://htmlcheatsheet.com/css/

browser history

The history of modern browsing began back in December of 1990 with the release of WorldWideWeb browser. It was written by Tim Berners-Lee while working for the European nuclear research agency known as CERN. It was later renamed to Nexus, to avoid confusion with the World Wide Web. Nexus was the first of its kind, and allowed users to view basic style sheets, read newsgroups, and even had spellcheck!
The release of Nexus was just the beginning though, as in the next decade people witnessed the first releases of browsers such as Mosaic Browser, which quickly gained popularity and became the most popular browser on the globe. From there, the growth of the World Wide Web exploded with the releases of Opera and Netscape Navigator browsers.

In 1995 the world got introduced to the first version of Internet Explorer, which became the dominant player in the market. At some point, Internet Explorer was used by more than 90% of all users. To counter this dominance, Netscape launched what would become Mozilla Foundation which develops and maintains Firefox. Soon after that, in 2003, Apple launched Safari, and in 2008, Google launched Chrome. There is a lot of competition among browsers still to this day, even though Chrome (and Chromium) is the dominant player in the market.

browser compatibility

it is impossible to imagine the Web without the use of browsers. We have witnessed a shift from standalone applications to HTML5 and Progressive Web Apps that allow applications to fully function within a browser. As companies compete for market share, different browsers are using different engines to display information on web pages. For example, Chrome and Chromium utilize Blink, while Safari uses WebKit. For your web development projects to have a broader reach, you must make sure that you're testing your web applications against browsers which are most likely to be used by users. Chrome, Safari, Firefox, and other Chromium-based browsers (Microsoft Edge, Brave, etc.) are more common among regular users. As exciting as it is to implement new features, there is a risk of rushing. "Can I Use" is a great resource to help you validate if new features are supported by browsers. It provides statistics on which browsers and platforms are supporting new technologies, and even which versions of those browsers support specific features.

As you're developing your applications, you must also consider whether your application should be fully mobile compatible. There are a couple of specifics about mobile browsers that you need to keep in mind.

1. On iOS and iPadOS, Safari is technically the only supported browser. Yes, you can install Chrome or Firefox, and you can even set them as a default, but they are not full browsers. They are still using the Safari rendering engine (WebKit). Therefore, for your web application to work for Apple users, you have to ensure support for WebKit and other technologies used in Safari. It's important to remember that mobile browsers are not one-to-one with their desktop counterparts. A project that works in the desktop version of Safari might still need adjustments to work properly on the mobile version of the same browser.
2. Another consideration for mobile browsers is the magnitude of different screen sizes. It is virtually impossible to have every physical device available to test, and thankfully browsers provide a way to emulate other devices. The important piece to remember is that when, for example, you emulate an iPhone in Chrome, all that you're emulating is the screen size. Keep in mind that any specific considerations of the operating system will not be reproducible. Which means that even though everything functions well in Chrome when emulating a device, it could behave differently on the actual phone or tablet device.

https://adactio.com/journal/17428

https://taskade.medium.com/history-of-web-browsers-the-evolution-of-digital-productivity-%EF%B8%8F-28fa2d4130fb

javascript base

External Linking

```
<script src="javascript.js"></script>
```

Internal Linking

```
<script>
  // Your JavaScript goes here!
  console.log("Hello, World!")
</script>
```

console.log() gives the output in developer console in inspect elements

```
let name = "John";
let surname = "Doe";

console.log(name);
console.log(surname);
```

```
let age = 11;
console.log(age); // outputs 11 to the console

age = 54;

console.log(age); // what will be output now?
```

let is used for initialisation of variables

```
const pi = 3.14;
pi = 10;

console.log(pi); // What will be output?
```

const is let, if we don't want variable reassignment

there is also **var**, similar to let and const used in old code
javascript is **dynamically typed**, so variables aren't bound to a single data type
a variable can be a string then be reassigned a integer.

It has few datatypes, like **bigInt** for numbers larger than 2^{53}-1 (+/-ve) with a **n at end**

```
// the "n" at the end means it's a BigInt
const bigInt = 1234567890123456789012345678901234567890n;
```

strings are surrounded by quotes

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
```

single and double quotes serve same purpose while **backticks** are used for variable embed

boolean values for **true;** & **false;** and comparisons

```
let nameFieldChecked = true; // yes, name field is checked
let ageFieldChecked = false; // no, age field is not checked

let isGreater = 4 > 1;
```

null

```
let age = null;
```

undefined meaning variable is not assigned yet

```
let age = 100;

// change the value to undefined
age = undefined;
```

primitive datatypes store single value while **object** datatypes stores complex data or collections of datatypes & **symbol** datatype store unique identifiers for the objects.

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof 10n // "bigint"

typeof true // "boolean"

typeof "foo" // "string"

typeof Symbol("id") // "symbol"

typeof Math // "object"  (1)

typeof null // "object"  (2)

typeof alert // "function"  (3)
```

typeof returns type of operand (here math is a builtin object that provides math operations while the null being a object is a officially recognised error of typeof, not removed for backward compatibilty

strings

concatenation using +

```
const greeting = "Hello";
const name = "Chris";
console.log(greeting + ", " + name); // "Hello, Chris"
```

we can also concatenate numbers to strings, as number converts to a string in this case

even though backticks provide clean usage

```
const greeting = "Hello";
const name = "Chris";
console.log(`${greeting}, ${name}`); // "Hello, Chris"
```

backticks also allow mathematical expressions between variables in string

```
const song = "Fight the Youth";
const score = 9;
const highestScore = 10;
const output = `I like the song ${song}. I gave it a score of ${
  (score / highestScore) * 100
}%.`;
```

backticks, also called **template literals** respect newlines inside of string and will maintain it while we would need **\n** in normal strings.

backticks also help us use actual quotes in our strings without error

```
const goodQuotes1 = 'She said "I think so!"';
const goodQuotes2 = `She said "I'm not going in there!"`;
```

Number() and String() function

```
const myString = "123";
const myNum = Number(myString);
console.log(typeof myNum);
// number
```

```
const myNum2 = 123;
const myString2 = String(myNum2);
console.log(typeof myString2);
// string
```
\

string functions

```
let str = "JavaScript";

console.log(
  str.length,         // 10  -> String length
  str.charAt(1),      // "a" -> Character at index 1
  str.charCodeAt(0),  // 74  -> Unicode of 'J'
  str.at(-1),         // "t" -> Last character (ES2022+)
  str[2],             // "v" -> Character at index 2 (Bracket notation)
  str.slice(0, 4),    // "Java" -> Extract from index 0 to 3
  str.substring(4, 10), // "Script" -> Extract from index 4 to end
  str.substr(4, 6)    // "Script" -> start at 4, for 6 characters
);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String for all string methods.

basic comparison

```
alert( 2 > 1 );  // true (correct)
alert( 2 == 1 ); // false (wrong)
alert( 2 != 1 ); // true (correct)

let result = 5 > 4; // assign the result of the comparison
alert( result ); // true

alert( 'Z' > 'A' ); // true
alert( 'Glow' > 'Glee' ); // true
alert( 'Bee' > 'Be' ); // true
```

a is greater than A due to indexing in encoding of unicode used in js

```
alert( '2' > 1 ); // true, string '2' becomes a number 2
alert( '01' == 1 ); // true, string '01' becomes a number 1
```

javascript will convert these strings to numbers

```
let a = 0;
alert( Boolean(a) ); // false

let b = "0";
alert( Boolean(b) ); // true

alert(a == b); // true!
```

funny consequence of flexible type conversion

```
alert( 0 === false ); // false, because the types are different
```

strict equality checker verifies for type equality (== would return true) (!== strict unequality)

or (||)

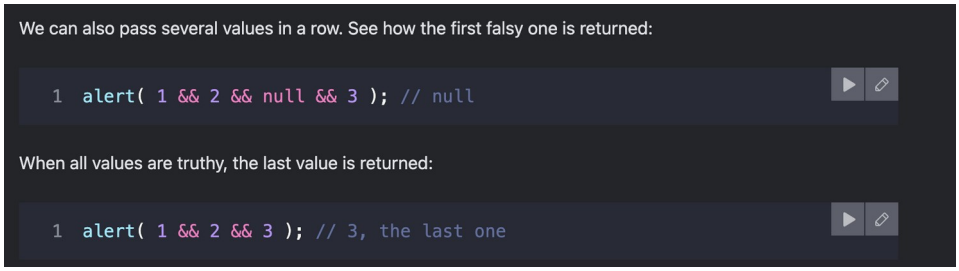
```
alert( true || true );   // true
alert( false || true );  // true
alert( true || false );  // true
alert( false || false ); // false
```

```
let firstName = "";
let lastName = "";
let nickName = "SuperCoder";

alert( firstName || lastName || nickName || "Anonymous");
```

In case of multiple ORs, if all are false, last value is returned, if any is true, first true value is returned

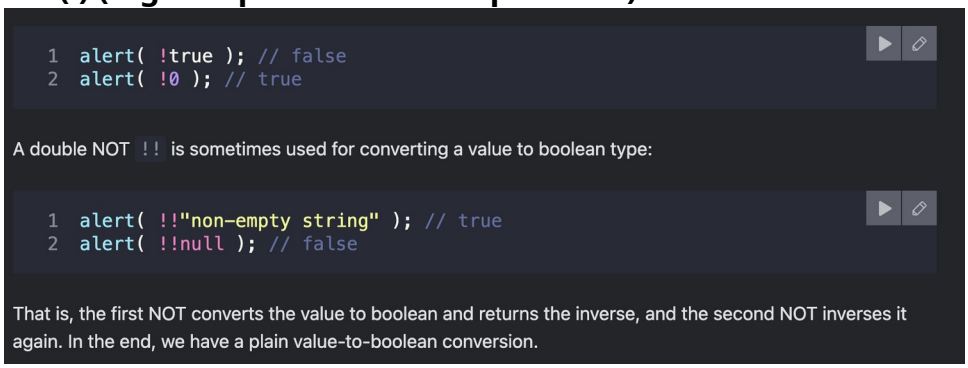
and (&&)

```
alert( true && true );   // true
alert( false && true );  // false
alert( true && false );  // false
alert( false && false ); // false
```



In case of multiple ANDs, if all are true, last value is returned, if any is false, first false value is returned

not(!) (highest precedence in operators)



simpler method is just using Boolean(“non-empty string”) or Boolean(null)

ternary operator (?)

```
condition ? expression_if_true : expression_if_false;
```

if else statement syntax

```
const randomValue = Math.random();

if (randomValue < 0.33) {console.log("Action 1: Low probability range (0 - 0.33)");}
else if (randomValue < 0.66) {console.log("Action 2: Medium probability range (0.33 - 0.66)");}
else {console.log("Action 3: High probability range (0.66 - 1)");}
```

switch statement syntax

```
let a = 2 + 2;

switch (a) {
  case 3:
    alert( 'Too small' );
    break;
  case 4:
    alert( 'Exactly!' );
    break;
  case 5:
    alert( 'Too big' );
    break;
  default:
    alert( "I don't know such values" );
}
```

```
let a = 3;

switch (a) {
  case 4:
    alert('Right!');
    break;

  case 3: // (*) grouped two cases
  case 5:
    alert('Wrong!');
    alert("Why don't you take a math class?");
    break;

  default:
    alert('The result is strange. Really.');
}
```

If we want case 3 and case 5 to have same response

```
let arg = prompt("Enter a value?");
switch (arg) {
  case '0':
  case '1':
    alert( 'One or zero' );
    break;

  case '2':
    alert( 'Two' );
    break;

  case 3:
    alert( 'Never executes!' );
    break;
  default:
    alert( 'An unknown value' );
}
```

prompt() always takes string as a input, case 3 can not execute ever

function syntax

```
function favoriteAnimal(animal) {
    return animal + " is my favorite animal!"
}

console.log(favoriteAnimal('Goat'))
```

default parameter

```
function hello(name = "Chris") {
  console.log(`Hello ${name}!`);
}

hello("Ari"); // Hello Ari!
hello(); // Hello Chris!
```

anonymous function assigned to variable as a object

```
 let greet = function(name) {
   console.log(`Hello, ${name}!`);
 };

 greet("Alice");
```

addEventListener is just a method, ultimately the function is passed without a name

```
document.getElementById("myButton").addEventListener("click", function () {
  console.log("Button was clicked!");
});
```

we can also use **arrow function**

```
textBox.addEventListener("keydown", (event) => {
  console.log(`You pressed "${event.key}".`);
});
```

we can even remove the parenthesis around event, in case of one parameter

```
document.getElementById("myButton").addEventListener("click", () => {
  console.log("Button was clicked!");
});
```

```
const originals = [1, 2, 3];

const doubled = originals.map(item => item * 2);

console.log(doubled); // [2, 4, 6]
```

with only one return statement in function, we can even remove the {} & write the return code in parameter brace itself(map() function takes all elements one at a time)

map() function

map() function **takes a function as an argument**, and applies it to every array element and returns a new array without modifying the original array.

```
function addOne(num) {
  return num + 1;
}
const arr = [1, 2, 3, 4, 5];
const mappedArr = arr.map(addOne);
console.log(mappedArr); // Outputs [2, 3, 4, 5, 6]
```

```
const arr = [1, 2, 3, 4, 5];
const mappedArr = arr.map((num) => num + 1);
console.log(mappedArr); // Outputs [2, 3, 4, 5, 6]
```

filter() function

filter() function **also takes a function as an argument,** but only returns the array values that are true for the function in argument.

```
unction isOdd(num) {
  return num % 2 !== 0;
}
const arr = [1, 2, 3, 4, 5];
const oddNums = arr.filter(isOdd);
console.log(oddNums); // Outputs [1, 3, 5];
console.log(arr); // Outputs [1, 2, 3, 4, 5], original array is
    not affected
```

reduce() function

```
const arr = [1, 2, 3, 4, 5];
const productOfAllNums = arr.reduce( (total, currentItem) => {return total * currentItem;}, 1);
console.log(productOfAllNums); // Outputs 120;
console.log(arr); // Outputs [1, 2, 3, 4, 5]
```

here 1 is the initial value of the first argument of first argument of reduce function, which is the accumulator in this case, while currentItem will be the array items being iterated. It

function scope

HTML

```
<!-- Excerpt from my HTML -->
<script src="first.js"></script>
<script src="second.js"></script>
<script>
  greeting();
</script>
```

JS

```
// first.js
const name = "Chris";
function greeting() {
  alert(`Hello ${name}: welcome to our company.`);
}
```

JS

```
// second.js
const name = "Zaptec";
function greeting() {
  alert(`Our company is called ${name}.`);
}
```

The functions declared in both js files have global scope in the html file.
functions declared in block scopes, end their scope after {} ends for example, if a function is declared in a if {} block, then it can't be accessed later until its assigned to a variable with larger accessible scope

The second js file will not load because name is declared twice, "Hello Chris ...." will run

If we remove the const name from second js, the “Our company is called Chris” will run because functions can be declared twice, latest definition will execute

```
function showPrimes(n) {

  for (let i = 2; i < n; i++) {
    if (!isPrime(i)) continue;

    alert(i);  // a prime
  }
}

function isPrime(n) {
  for (let i = 2; i < n; i++) {
    if ( n % i == 0) return false;
  }
  return true;
}
```

functions can be used, even when declared later due to movement to top scope before execution, variables assigned to functions, cannot be called before declared as they aren’t moved to the top scope.

javascript used a call stack to execute its functions, the function being called is put over to the top of the stack and execution begins from the top till stack is empty. If the function is calling another function, the other function is put on top of the stack and executed first and then proceeded downwards. The order of declaration does not matter as stack will always have all the functions in right order for complete execution of program

array

```
const cars = ["Saab", "Volvo", "BMW"];
```

most preferred initialisation

```
const cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

also work for initialisation

```
let arr1 = new Array(); // Creates an empty array
let arr2 = new Array(5); // Creates an array with 5 empty slots
let arr3 = new Array(1, 2, 3); // Creates an array with elements [1, 2, 3]
```

avoid due to ambiguity in (5) and (1,2,3)

```
// Step 1: Create an array
let fruits = ["Apple", "Banana", "Cherry"];

// Step 2: Assign an element to a variable
let myFruit = fruits[1]; // "Banana"

// Step 3: Change the variable
myFruit = "Mango";

console.log(myFruit); // Output: "Mango"
console.log(fruits[1]); // Output: "Banana" (Original array remains unchanged)
```

accessing and changing elements

```
const person = {firstName:"John", lastName:"Doe", age:46};
document.getElementById("demo").innerHTML = person.firstName;
```

arrays are also objects, person.firstName will return “John”, also the document get element thing just finds a html element with id “demo” and outputs the js output in html site.
You can have objects, functions and even arrays in arrays.

read about all useful array methods mentioned here
https://www.w3schools.com/js/js_array_methods.asp

iterating an array by for looping / map() / filter()

```
1   // Step 1: Create an array
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
3
4   // Step 2: Use a for loop to print each number
5   console.log("For Loop Output:");
6   for (let i = 0; i < numbers.length; i++) {
7       console.log(numbers[i]);
8   }
9
10  // Step 3: Use map() to access each elemement
11  let squaredNumbers = numbers.map(num => num * num);
12  console.log("Mapped Array (Squared Numbers):", squaredNumbers);
13
14  // Step 4: Use filter() to get only even numbers
15  let evenNumbers = numbers.filter(num => num % 2 === 0);
16  console.log("Filtered Array (Even Numbers):", evenNumbers);
17
```

The Difference Between Arrays and Objects

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**.

```
const person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
person.length;    // Will return 3
person[0];        // Will return "John"
```

nested array & accessing each element

```
const myObj = {
  name: "John",
  age: 30,
  cars: [
    {name:"Ford", models:["Fiesta", "Focus", "Mustang"]},
    {name:"BMW", models:["320", "X3", "X5"]},
    {name:"Fiat", models:["500", "Panda"]}
  ]
}
for (let i in myObj.cars) {
  x += "<h1>" + myObj.cars[i].name + "</h1>";
  for (let j in myObj.cars[i].models) {
    x += myObj.cars[i].models[j];
  }
}
```

while loop

```
const cats = ["Pete", "Biggles", "Jasmine"];

let myFavoriteCats = "My cats are called ";

let i = 0;

while (i < cats.length) {
  if (i === cats.length - 1) {
    myFavoriteCats += `and ${cats[i]}.`;
  } else {
    myFavoriteCats += `${cats[i]}, `;
  }

  i++;
}

console.log(myFavoriteCats); // "My cats are called Pete, Biggles, and Jasmine."
```

break and continue

```
let sum = 0;

while (true) {

  let value = +prompt("Enter a number", '');

  if (!value) break; // (*)

  sum += value;

}
alert( 'Sum: ' + sum );
```

```
for (let i = 0; i < 10; i++) {

  // if true, skip the remaining part of the body
  if (i % 2 == 0) continue;

  alert(i); // 1, then 3, 5, 7, 9
}
```

breaking out of a whole nested loop at once using label names before for loops

```
outer: for (let i = 0; i < 3; i++) {

  for (let j = 0; j < 3; j++) {

    let input = prompt(`Value at coords (${i},${j})`, '');

    // if an empty string or canceled, then break out of both loops
    if (!input) break outer; // (*)

    // do something with the value...
  }
}

alert('Done!');
```

document object model

When your HTML code is parsed by a web browser, it is converted to the DOM, as was mentioned above. One of the primary differences is that these nodes are JavaScript objects that have many properties and methods attached to them.

Query selectors

- `element.querySelector(selector)` - returns a reference to the first match of *selector*.
- `element.querySelectorAll(selectors)` - returns a “NodeList” containing references to all of the matches of the *selectors*.

relational selectors

```
const container = document.querySelector("#container");

// selects the first child of #container => .display
const display = container.firstElementChild;
console.log(display);  // <div class="display"></div>
const controls = document.querySelector(".controls");

// selects the prior sibling => .display
const display = controls.previousElementSibling;
console.log(display); // <div class="display"></div>
```

displaying already existing DOM objects / html elements in console

creating new elements in javascript

document.createElement(tagName, [options]) creates a new element of type tag given while options means we can add more parameters in the function. This **doesn't** add the element to DOM yet. **document.parentNode.appendChild(childNode)** will add the childnode as the last node of parent mentioned.

```
const newDiv = document.createElement("div"); // Creates a <div> element

document.body.appendChild(newDiv); // Adds it at the end of <body>
```

document.parentNode.insertBefore(newNode, referenceNode)
inserts newNode in the parent node before referenceNode

document.parentNode.removeChild(newNode, referenceNode)
removes the child from parent and returns the reference of the child

lets say you have created a div element using createElement in the script
we can add css and other styles to this

div.style.color = "blue";

div.style.cssText = "color: blue; background: white;";

div.style.property adds the attribute to the set of styles already mentioned into inline html

so it ends up being modified to , which is more precedent than css styles, unless css style has **!important** after the attribute.

```
#box {
  color: blue !important;
}
```

that being said, you can override css's !important with a !important in your js code.

```
box.setAttribute("style", "color: red !important;"); // Forces red
```

as we can see above, we also have div.setAttribute("style", "color: blue; background: white;"); this replaces & removes the inline style mentioned in html with whatever is mentioned in js code. div.style.cssText also removes the inline style mentioned in html.

For a kebab cased property like background-color, we **cannot** use dot notation, div.style.background-color; will just subtract color from background and give error

we either use camel case div.style.backgroundColor = "yellow"; or just remove the whole inline style with either two methods

```
div.style.cssText = "background-color: yellow;";
```

```
div.setAttribute("style", "background-color: yellow;");
```

javascript Copy Edit

```
div.setAttribute("id", "theDiv");
```

✅ What happens?

- If `div` already has an `id`, it gets updated to `"theDiv"`.
- If `div` doesn't have an `id`, a new `id` attribute is created and set to `"theDiv"`.

Example Before Execution:

html Copy Edit

```
<div></div>
```

Example After Execution:

html Copy Edit

```
<div id="theDiv"></div>
```

```javascript
div.getAttribute("id");
```

✅ **What happens?**

- This fetches the **value** of the `id` attribute.
- If `id="theDiv"`, it returns `"theDiv"`.
- If there is **no** `id`, it returns `null`.

Example:

```javascript
console.log(div.getAttribute("id")); // Outputs: "theDiv"
```

```javascript
div.removeAttribute("id");
```

✅ **What happens?**

- It **completely removes** the `id` attribute from the element.
- The element will no longer have an `id` at all.

Example After Execution:

```html
<div></div>
```

```javascript
div.classList.add("new");
```

✔️ **What happens?**

- If `div` **does not** have the class `"new"`, it gets **added**.
- If `div` **already has** `"new"`, nothing changes (no duplicates).

Example Before Execution:

```html
<div class="box"></div>
```

After Execution:

```html
<div class="box new"></div>
```

javascript Copy Edit

```
div.classList.remove("new");
```

✓ **What happens?**

- If `div` **has** the class `"new"`, it gets **removed**.
- If it **doesn't have** `"new"`, nothing happens.

Example Before Execution:

html Copy Edit

```
<div class="box new"></div>
```

After Execution:

html Copy Edit

```
<div class="box"></div>
```

javascript Copy Edit

```
div.classList.toggle("active");
```

✓ **What happens?**

- If `div` **does not** have `"active"`, it **adds** it.
- If `div` **already has** `"active"`, it **removes** it.

It is often standard (and cleaner) to toggle a CSS style rather than adding and removing inline CSS.

Before Execution:

html Copy Edit

```
<div id="example">Hello, <strong>World!</strong></div>
```

JavaScript:

javascript Copy Edit

```
document.getElementById("example").textContent = "Hello, Universe!";
```

After Execution:

html Copy Edit

```
<div id="example">Hello, Universe!</div>
```

replaces all the html content with plain text, does not interpret html so even if we have a or inside the textContent, it will not be parsed. It also works with div.textContent = “Hello, Universe!”;

replaces all the html content with plain text, but the html is interpreted if mentioned inside innerHTML. It also works with div.innerHTML = “Hello, Universe!”;

DOM example

```
<!-- your HTML file: -->
<body>
  <h1>THE TITLE OF YOUR WEBPAGE</h1>
  <div id="container"></div>
</body>
```

```
// your JavaScript file
const container = document.querySelector("#container");

const content = document.createElement("div");
content.classList.add("content");
content.textContent = "This is the glorious text-content!";

container.appendChild(content);
```

```
<!-- The DOM -->
<body>
  <h1>THE TITLE OF YOUR WEBPAGE</h1>
  <div id="container">
    <div class="content">This is the glorious text-content!</div>
  </div>
</body>
```

events

events are actions that occur on a webpage, such as mouseclicks & keypresses, javascript helps us listen and react to these events.

we can specify the reaction in HTML itself, using <button> element <button onclick="alert('Hello World')">Click Me</button> this however clutters our HTML with js and restricts us to one onclick propery per DOM element or we can simply give the button element a id & implement further js in the script part of the code

```html
<!-- the HTML file -->
<button id="btn">Click Me</button>
```

```js
// the JavaScript file
const btn = document.querySelector("#btn");
btn.onclick = () => alert("Hello World");
```

we use a arrow function here, but it would be the same as

```js
function showAlert() {
    alert("Hello World");
}
btn.onclick = showAlert;
```

both of the methods mentioned above, restrict us to one reaction per event. To bypass that we must use addEventListener functions in our js code.

```html
<!-- the HTML file -->
<button id="btn">Click Me Too</button>
```

```js
// the JavaScript file
const btn = document.querySelector("#btn");
btn.addEventListener("click", () => {
  alert("Hello World");
});
```

we can add multiple addEventListener functions to same button element based on how many reactions we want like from alerting information to outputting to console.log

we can use user created functions for event reactions as well

```html
<!-- the HTML file -->
<!-- METHODS 2 & 3 -->
<button id="btn">CLICK ME BABY</button>
```

```js
// the JavaScript file
// METHODS 2 & 3
function alertFunction() {
  alert("YAY! YOU DID IT!");
}
const btn = document.querySelector("#btn");

// METHOD 2
btn.onclick = alertFunction;

// METHOD 3
btn.addEventListener("click", alertFunction);
```

e parameter object

```js
btn.addEventListener("click", function (e) {
  console.log(e);
});
```

e is a object that js automatically provides, when an event occurs. It contains various details about the event like what key was pressed, what event ,where the mouse is and if ctrl or shift was also pressed during the event. There is nothing special about e, the function in second part of event listener takes an event object by definition and event handler simply passes the reference to the event occurred.

we can use the inner information of e object, inside the function

```js
btn.addEventListener("click", function (e) {
  console.log(e.target);
});
```

In this case, e.target refers to the button element being clicked

```js
btn.addEventListener("click", function (e) {
  e.target.style.background = "blue";
});
```

and this changes the background of button element being clicked

adding event listeners to many events at the same time (reaction to every event in one function)

```html
<div id="container">
  <button id="one">Click Me</button>
  <button id="two">Click Me</button>
  <button id="three">Click Me</button>
</div>
```

```js
const buttons = document.querySelectorAll("button");

// we use the .forEach method to iterate through each button
buttons.forEach((button) => {
  // and for each one we add a 'click' listener
  button.addEventListener("click", () => {
    alert(button.id);
  });
});
```

Here the querySelectorAll returns a node list which is somewhat similar to an array.

https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/Events#event_objects

objects

objects are used to store keyed data and more complex entities. we can create object by

```
let user = new Object(); // "object constructor" syntax
let user = {};  // "object literal" syntax
```

```
let user = {     // an object
  name: "John",  // by key "name" store value "John"
  age: 30        // by key "age" store value 30
};
```

we can add & then later remove new properties anytime, even after declaration of object

```
user.isAdmin = true;

delete user.age;
```

we can also use multi worded property key names, but they must be in quotes &
we must use a different notation to access the value, as you cant put spaces in dot notation

```
"likes birds": true  // multiword property name must be quoted
```

```
// get
alert(user["likes birds"]); // true
```

one more advantage of bracket notation is that it provides us with the ability to use variables that be defined in the code or via user input, for specific output (not possible with dot notation)

```
let user = {
  name: "John",
  age: 30
};

let key = prompt("What do you want to know about the user?", "name");

// access by variable
alert( user[key] ); // John (if enter "name")
```

Here prompt is just a input function

```
let result = prompt(message, defaultValue);
```

important examples for diverse usage and peculiar rules

```
let fruit = prompt("Which fruit to buy?", "apple");

let bag = {
  [fruit]: 5, // the name of the property is taken from the variable fruit
};

alert( bag.apple ); // 5 if fruit="apple"
```

```
let fruit = 'apple';
let bag = {
  [fruit + 'Computers']: 5 // bag.appleComputers = 5
};
```

```
function makeUser(name, age) {
  return {
    name: name,
    age: age,
    // ...other properties
  };
}

let user = makeUser("John", 30);
alert(user.name); // John
```

```
let user = {
  name,  // same as name:name
  age: 30
};
```

```
let user = { name: "John", age: 30 };

alert( "age" in user ); // true, user.age exists
alert( "blabla" in user ); // false, user.blabla doesn't exist
```

```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let key in user) {
  // keys
  alert( key );  // name, age, isAdmin
  // values for the keys
  alert( user[key] ); // John, 30, true
}
```

```
let codes = {
  "49": "Germany",
  "41": "Switzerland",
  "44": "Great Britain",
  // ..,
  "1": "USA"
};

for (let code in codes) {
  alert(code); // 1, 41, 44, 49
}
```

object variables store a reference to the main object, this makes sense in the DOM as well, where we change the object in javascript, but the change is seen in the DOM object.

```
const obj = { data: 42 };

const objCopy = obj;

objCopy.data = 43;

console.log(obj); // { data: 43 }
console.log(objCopy); // { data: 43 }
```

```
function increaseCounterObject(objectCounter) {
  objectCounter.counter += 1;
}

function increaseCounterPrimitive(primitiveCounter) {
  primitiveCounter += 1;
}

const object = { counter: 0 };
let primitive = 0;

increaseCounterObject(object);
increaseCounterPrimitive(primitive);
```

Copy

the object counter would increase by 1, and the primitive counter wouldn't change

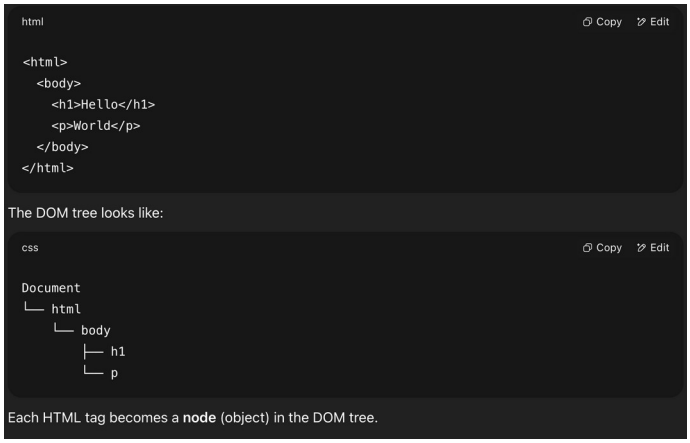
the parameter objectCounter contains a reference to the same object as the object variable we gave it, while primitiveCounter only contains a copy of the primitive value.

javascript developer tool resource

https://www.theodinproject.com/lessons/foundations-javascript-developer-tools

DOM Tree

The DOM is a **programming interface for web documents**. It represents the structure of a webpage as a **tree of objects** that can be accessed and manipulated using programming languages like JavaScript. DOM is helpful in increasing interactivity because it lets us add / remove / modify existing elements/nodes of the DOM tree directly from javascript.



nvm

The NVM is a tool that lets you install & manage version of node.js on your system
https://www.theodinproject.com/lessons/foundations-installing-node-js
after installation, **node** command opens js runtime environment, **.exit** to exit the environment

npm

npm is a package manager, a gigantic repository of plugins, libraries & other tools which can be installed locally & used in our own files. npm already gets installed when we install nvm.

every project will have a **package.json** file which contains information about the project, such as its name & dependencies required to run the file. these files cannot be run in go live ports as they only work for vanilla stack projects. we would need to go into the project file from terminal and type **npm install**

now **npm run dev** will run the app in http://localhost:3000/

npm scripts

```
{
  // ... other package.json stuff
  "scripts": {
    "build": "webpack",
    "dev": "webpack serve",
    "deploy": "git subtree push --prefix dist origin gh-pages"
  },
  // ... other package.json stuff
}
```

npm scripts defined in package json file can act as simple keywords for large commands
npm run deploy will run **git subtree push --prefix dist origin gh-pages**

export / import & ES6 Modules

we can export/import functions/variables from one file to another for use.
there are two types of exports; named exports for importing multiple files but every file can only export default one thing so when we import anything from that file, the code knows what it should import but this import can now be named anything in new file. export default can be written in the end of component file or just during declaration

```
<script src="two.js" type="module"></script>
```

we can also export a complete file in the form of a **ES6 Module**, if two.js was importing something from one.js and three.js, then those files will be automatically loaded in the module as well making the whole process easier.

webpacks

webpack is a **module bundler** for JavaScript applications. It takes all your JS, CSS, images, fonts, etc., and **bundles** them into optimized assets that the browser can use & import/export efficiently.

- It uses **entry points** to know where to start bundling.
- It applies **loaders** to handle different file types (.js, .css, .png, etc.).
- It uses **plugins** for advanced tasks like minification, environment variables, etc.
- It outputs a final **bundle** (or multiple) which is served to users.
-

json (javascript object notation)

```
{
  "name": "Varun",
  "age": 23,
  "skills": ["JavaScript", "C++", "React"]
}
```

JSON is a lightweight data format used to store and exchange data.
It stores data like javascript objects are stored & if u run it on a jS runtime with a variable name given and output using the dot notation, you will get the output like you would with a javascript object. it is used in config files like package.json, the frontend and backend contact each other using JSON files & No SQL databases store information also in JSON files.

linting

It is the process of automatically checking your code for errors and may even fix them.
ESLint checks for code errors & inconsistent formats & **Prettier** checks for code readability.

ES6 & Babel

javascript is a language that conforms to ecmascript standard, **ES6** is the version of JS released in 2015 for public use. **Babel** is a tool that converts future javascript versions to

older javascript versions so browsers don't have compatibility can still run it, it is pre installed on next.js applications and create-react-apps.

asynchronous code

javascript runs on a single thread so it can only do one thing at a time, a slow task can block everything else. To avoid this situation, the program can move ahead and start future tasks without waiting for previous task to finish. A **callback** is a function that you pass as an argument to another function, the whole idea is that one function can only run when another function has been executed to combat the asynchronous code vulnerabilities in javascript, that being said, heavy usage of callbacks can make the code hard to read, maintain & debug called **callback hell.**

A **promise function** is another cleaner way to ensure that a line of code is only executed after another process is completed, kind of like a promise for example if a function is getting data from backend and polishing it to return it to be used for some purpose like extracting a key to value pair or using it, we can use promise function to ensure that those lines of code are only executed once the data is procured & the program does not move ahead to execute it. **async functions** are just syntactical sugar for promises, a function that uses **await** in its body which waits for a line of code to complete like **fetching** data from server or api with HTTP requests(webpage requesting from server/api) before moving forward must be declared **async.**

data validation using js

https://www.theodinproject.com/lessons/node-path-javascript-form-validation-with-javascript

API

Application Programming Interfaces are set of rules that lets software talk to other softwares, in context of web APIs, these allows websites and webapplications to talk to other server like backend or online servers to fetch/httprequest/give information without knowing fully how the server really works. **CORS** is a security features that manages the sharing of data between two different origins like your webpage and some other server, it recquires some setup and other complexities to use.

https://www.theodinproject.com/lessons/node-path-javascript-working-with-apis

what is react & next.js

react is a javascript library for building interactive user interface components easily as it has many already written helpful functions (API) & leaves it up to developer in their use method.

next.js is a react framework that gives you building blocks to create web applications by providing tools + configurations needed in react & additional structure, features and

optimisations. react is use to build the UI & then next.js is adopted to solve common app requirements like routing, data fetching & caching.

react setup

npm create vite@latest my-first-react-app -- --template react

cd my-first-react-app

npm install

npm run dev

localhost:5173 will now host your react application

react components

components are functions written in jsx format that are reusable piece of code that return some element that can be used like a navbar or home dashboard in many webpage files the function itself **must be capitalized** & must be exported & imported for use in main file.

there is **index.html** where the html of main page exists; there is **main.jsx** where we are taking an element from html by their id & adding components to them from javascript and a **index.css** for the css that applies on all elements in index.html and subsequently whatever main.jsx is pulling from index.html. if we import a component like **App.jsx** then it can have its own styles in **App.css**

one important thing to understand is that later declarations in css always stand so if we import index.css in main.jsx and then import App.jsx which imports App.css in its own first line, then App.css style is maintained over index.css style rules.

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'
import Greeting from './Greeting.jsx'


createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App/>
  </StrictMode>,
)
```

JSX

JSX is just javascript; everything outside of return statement or inside {} in return statements are javascript while remaining things in return statement look like html but they are just syntactical sugar and will be converted into React.createElement form of javascript.
It has three rules

1) a jsx component can only return one element; to return multiple u must wrap them in one container

2) close all tags even tags like

3) camelCase is used mostly because you cant really use dashes since this is javascript and names like class are reserved already so we must use className.

good example to convert a html to jsx usable html code
https://www.theodinproject.com/lessons/node-path-react-new-what-is-jsx

react props

props is a function argument, when we call the function, we can give few pairs of keys and values that will belong to the function called as a an object argument with key – values.

<functionname key1={value1} key2={value2}/> syntax for function call would mean that if the function ever called props.key1 in initialization then value1 will replace it and be used in its place.

```jsx
function ListItem(props) {
  return <li>{props.animalname}</li>
}

function List(props) {
  return (
    <ul>
      {props.objectkey.map((animaliteration) => {
        return <ListItem key={animaliteration} animalname={animaliteration} />;
      })}
    </ul>
  );
}

function App() {
  const animalslist = ["Lion", "Cow", "Snake", "Lizard"];

  return (
    <div>
      <h1>Animals: </h1>
      <List objectkey={animalslist} />
    </div>
  );
}
```

when we map a array or list and output it to a html list, we must provide a **specific key** as given in code above for every list item, because if the list were to change, then react needs to just apply minimal real dom operations to update the UI. The argument in a function using **(props)** can also just be **({keyname1, keyname2, keyname3})** for helpful readability, this is called **prop destructuring** https://www.theodinproject.com/lessons/node-path-react-new-passing-data-between-components

react state

react has special functions that let you hook into react's features in a functional component, called **hooks.** hooks must be declared at the first most line inside a component function & they cannot be declared in if, nested or conditional blocks.

one common hook is a **useState hook** that we can import from 'react'.

```jsx
import { useState } from 'react';
import viteLogo from '/vite.svg';
import './App.css';

function App() {
  const [count, setCount] = useState(10);

  return (
    <>
      <div>The count is {count}</div>
      <button onClick={() => { setCount(count + 1); }}>
        Update count
      </button>
    </>
  );
}
```

const [variable, variable updating function] = useState(variable's initial value);

we may now declare a variable updating function later on;

the point of react useState is that unlike normal JS webpages that are rendered once; if an component is to change its state value, the component itself will be rerendered without refreshing the whole webpage, indicating the change. If this whole count function was written for a normal javascript variable, then the value will still be incremented but the change will not be reflected until the site is refreshed.

update the value of variable only using the variable updating function because the site will not rerender the component if it does not detect the setting variable updating function.

https://www.theodinproject.com/lessons/node-path-react-new-more-on-state

react effect

react effects helps us synchronize our code when state values are changed.

lets say we want to tell the time a user has stayed on the site in seconds

running a setInterval inbuilt function in javascript that runs every 1000 ms

if we just use state to manage the seconds, every time the webpage rerenders the, a new setInterval is started, giving us multiple intervals running at the same time and each incrementing the count of seconds passed. useEffect ensured the interval only runs once as it runs after the rendering.

```
import { useEffect, useState } from "react";

export default function Clock() {
  const [counter, setCounter] = useState(0);

  useEffect(() => {
    const key = setInterval(() => {
      setCounter(count => count + 1)
    }, 1000);

    return () => {
      clearInterval(key);
    };
  }, [])

  return (
    <p>{counter} seconds have passed.</p>
  );
}
```

we have a clearInterval cleanup function because in development of react applications, the strict mode renders each component two times causing two intervals that can be avoided by cleaning it during unmounting (mounting is creation of a react component in the DOM and unmounting is removing it from the DOM). **arrow functions** => are used in stuff like count = count + 1 or functions being called because of async nature of javascript to ensure that these steps are done in a specific order and count is not incrementing some older previous value and resetting the count. useEffect must have a dependency matrix [] in itself which can be empty but if it has some variable or prop inside it, the interval will be set again when this variable or prop is changed

```
useEffect(
  () => {
    // execute side effect
    return () => {
      // cleanup function on unmounting or re-running effect
    }
  },
  // optional dependency array
  [/* 0 or more entries */]
)
```

class components and component lifecycle methods

prop in early versions of react, components were defined in classes and there were special methods to be used on these classes called component lifecycle methods. In context of modern react development and next.js, you can **build almost anything** with **functional components** & **react hooks**

https://www.theodinproject.com/lessons/node-path-react-new-class-based-components

https://www.theodinproject.com/lessons/node-path-react-new-component-lifecycle-methods

react testing
https://www.theodinproject.com/lessons/node-path-react-new-introduction-to-react-testing
https://www.theodinproject.com/lessons/node-path-react-new-mocking-callbacks-and-components

react proptype checking (typescript solves this issue)
https://www.theodinproject.com/lessons/node-path-react-new-type-checking-with-proptypes

react routing (next.js app routing makes it much easier to do)
https://www.theodinproject.com/lessons/node-path-react-new-react-router

fetching data to react website & setting custom hooks
https://www.theodinproject.com/lessons/node-path-react-new-fetching-data-in-react#managing-multiple-fetch-requests

context api for managing states between components
https://www.theodinproject.com/lessons/node-path-react-new-managing-state-with-the-context-api

useReducer hook
https://www.theodinproject.com/lessons/node-path-react-new-reducing-state

useRef, useCallback & useMemo hooks
https://www.theodinproject.com/lessons/node-path-react-new-refs-and-memoization

learn webdesign architecture and patterns
https://www.patterns.dev/

final problem your apps can do a lot of things already using server-side javascript but there is still no memory, your app forgets all the changes made every time site is refreshed. this is where we need the backend.

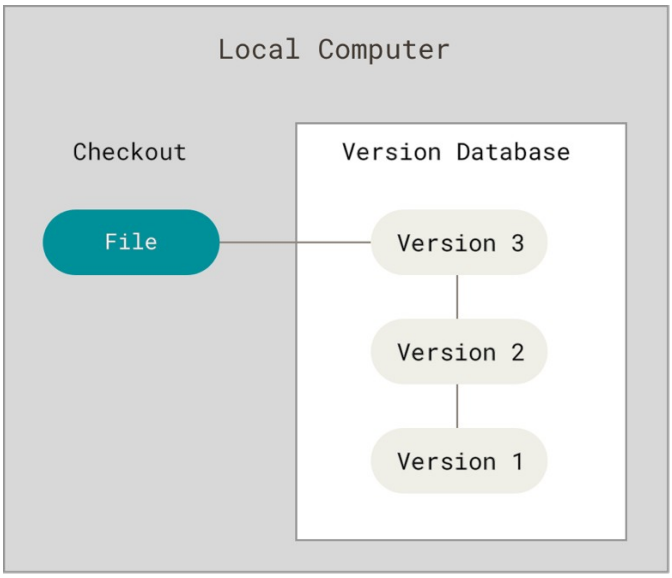
next.js dashboard documentation course
https://nextjs.org/learn/dashboard-app/getting-started

version control

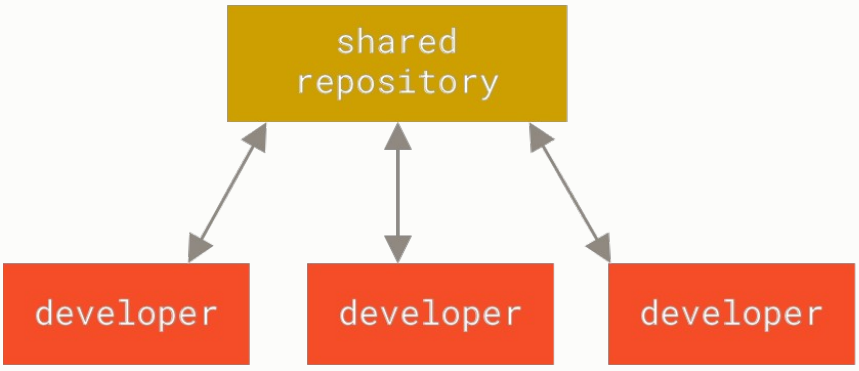
Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later which provides various advantages over just saving one result file.

The most simple version-control method of choice is to copy files into another directories perhaps with timestamps and push a different version on it. This is clever but is error prone

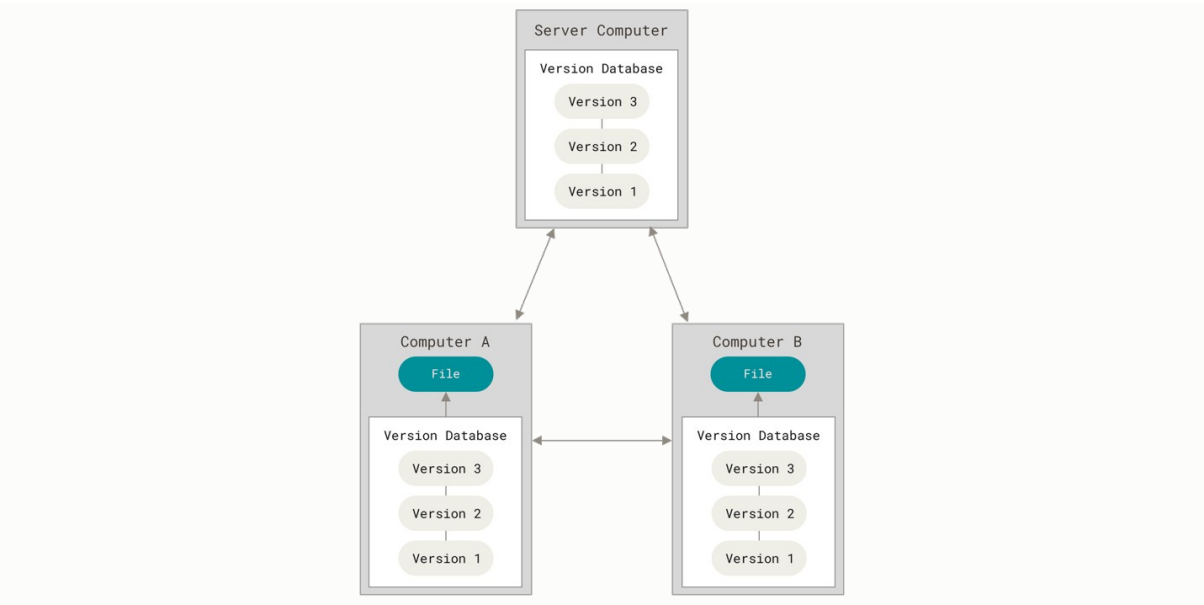
as its easy to forget which directory you are in and accidentally write to the wrong file or copy a different directory. A **Local Version Control System** combats this issue with a simple local database that keeps all changes to files under revision control and can recreate any file at any given timestamp.



To interact with other collaborators a **Centralised Version Control System** were developed which were a single server that contains all the versioned files and number of clients that check out the files. This helps all the developers have a rough idea of what everyone is working on and Administrators have fine-grained control over who can do what. The only point of failure rises from the possibility of server being down or hard disk of server ending up corrupted which could lead to disaster of epic proportions



Distributed Version Control Systems like **Git** solved the downsides of CVCS by giving the clients full mirrors of repository with every clone being a full backup of all data and multiple remote repositories where different groups can work in different ways on the same project.



the git

Git is a version control system that intelligently tracks changes in files.
Git is particularly useful when you and a group of people are all making changes to the same files at the same time as it keeps the historical record of each save. It used highly efficient compression algorithms to store your files and only saves the differences called **delta** and further compressed the delta. It stores everything in repository as objects and references the old objects in areas of code with no change and compresses these objects and deltas in pack files to further reduce storage overhead

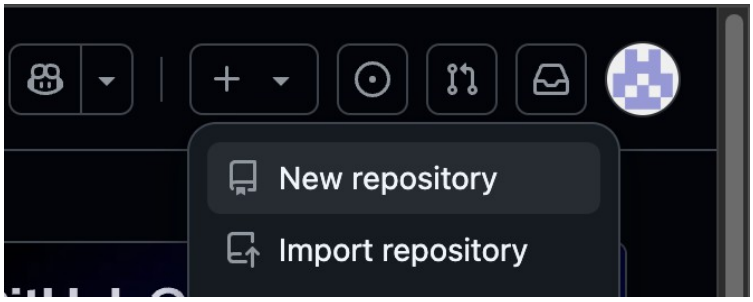
GitHub is a remote storage facility on the web for all your coding projects. When you upload files to GitHub, you'll store them in a "Git repository." This means that when you make changes (or "commits") to your files in GitHub, Git will automatically start to track and manage your changes. most people work on their files locally (on their own computer), then continually sync these local changes—and all the related Git data—with the central "remote" repository on GitHub.

Once you start to collaborate with others and all need to work on the same repository at the same time, you'll continually:

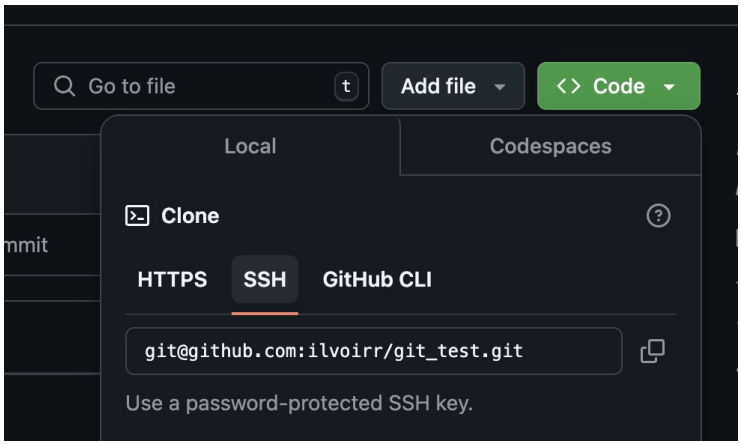
- **Pull** all the latest changes made by your collaborators from the remote repository on GitHub.
- **Push** back your own changes to the same remote repository on GitHub.

git workflow

create a Github Repository & Add a README.MD file for description of project.



get the local clone from SSH key to push commits from local server for others to see



git clone sshkey clones the repository to the local server in pwd

git remote -v gives full data on the remote online repositories that are linked to the cd

we can do any form of change to local repository like adding new folders or txtfiles

git status will return “untracked files” in **red** to indicate changes are not staged

git add foldername/txtfilename will add the change for staging and **git status** should return “changes to be committed” with files in green.

git add . will add all cd and subsequent inner directories to staging area

git commit -m “describe what the change is” should now commit to repository on your local machine completely and **git status** should now return “nothing to commit” and “your branch is ahead of origin/main by 1 commit” (origin is convention for remote repo you cloned and main is convention for the snapshot/branch of code on your local machine)

git add . && git commit -m “describe commit” allows both commands in 1 line

git commit opens default git editor, vim or visual studio code with **git config --global core.editor "code --wait"** with which you can commit with a subject + body, once you save the file, the commit is registered.

```
Add missing link and alt text to the company's logo

Screen readers won't read the images to users with disabilities
```

It is best practice to separate the subject (specifying the code’s action) and body (why the change is needed) with a blank/new line

git commit –-amend reset the staging area, so if we forgot to add & commit one file, we can add the forgotten file & then amending the commit will replace the older commit with new commit for pushing to main

git rebase -i HEAD~number of latest commits to change reset opens a text editor, where you can replace pick with reword & change their commit message to successfully change wrongfully written commit messages.

git reset related to removing latest commits but learn complexities from cgpt

git log returns all git entries you committed to the local repository

git push / git push origin main will upload your work to public github repository (in case its someone else’s repository you may have to push to a different snapshot or branch of code) and **git status** should show “your branch is upto date with origin/main”

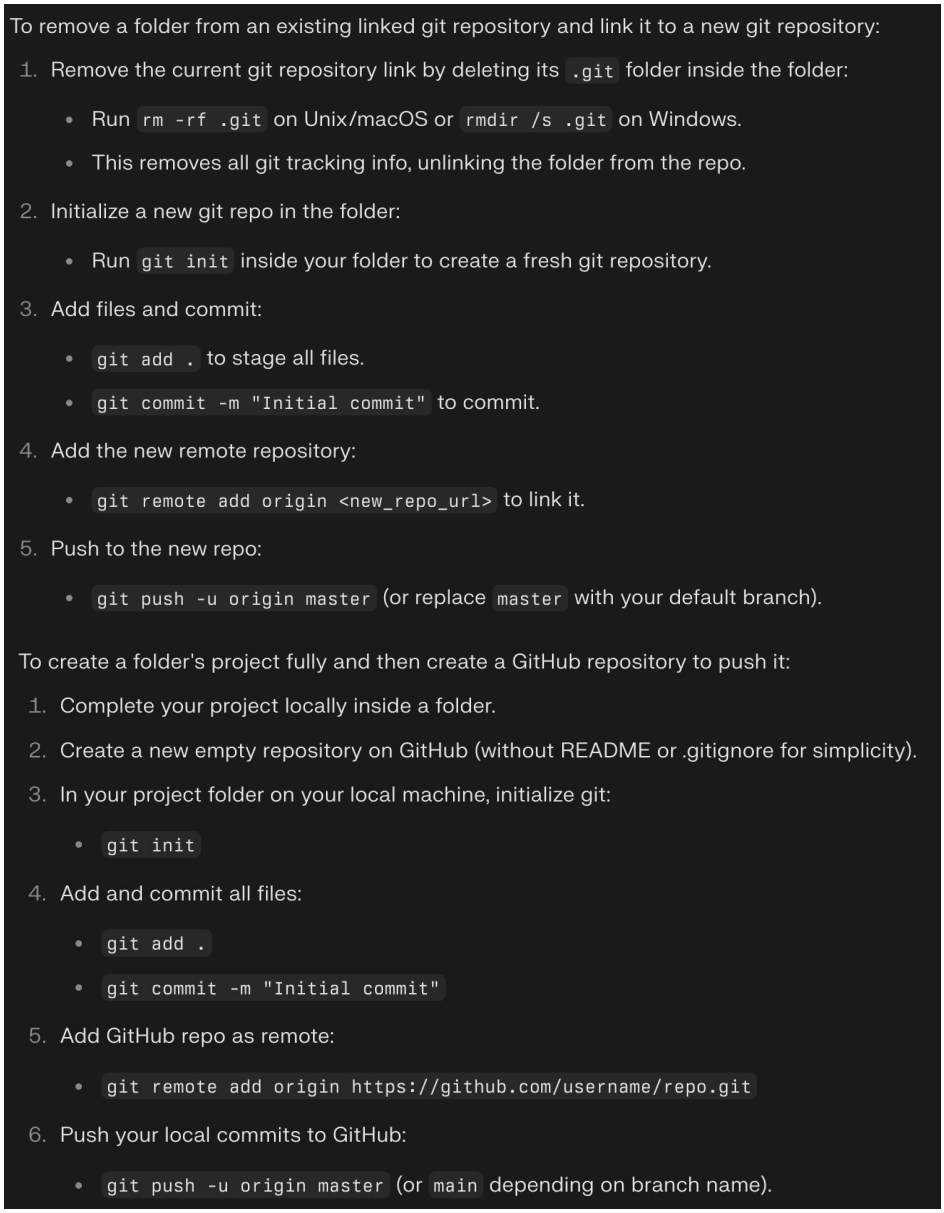
git pointers

HEAD pointer is pointing to the current snapshot of code that you are working on while the BRANCH pointer will initially point to the place where we created another strand/branch of code but as we write more commits in the new branch, it will ultimately go to next commit & point to the final tip in the new branch created. all commits in git, points to their parent commit.

https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet

contributing to real world repositories from your remote repo

https://www.theodinproject.com/lessons/javascript-using-git-in-the-real-world



To remove a folder from an existing linked git repository and link it to a new git repository:

1. Remove the current git repository link by deleting its `.git` folder inside the folder:
   - Run `rm -rf .git` on Unix/macOS or `rmdir /s .git` on Windows.
   - This removes all git tracking info, unlinking the folder from the repo.
2. Initialize a new git repo in the folder:
   - Run `git init` inside your folder to create a fresh git repository.
3. Add files and commit:
   - `git add .` to stage all files.
   - `git commit -m "Initial commit"` to commit.
4. Add the new remote repository:
   - `git remote add origin <new_repo_url>` to link it.
5. Push to the new repo:
   - `git push -u origin master` (or replace `master` with your default branch).

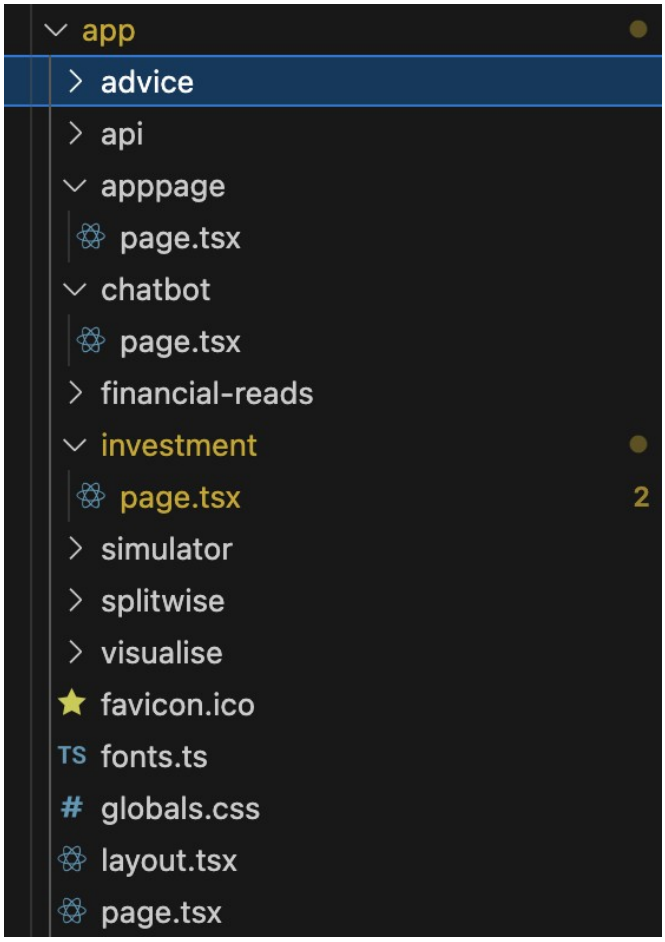
To create a folder's project fully and then create a GitHub repository to push it:

1. Complete your project locally inside a folder.
2. Create a new empty repository on GitHub (without README or .gitignore for simplicity).
3. In your project folder on your local machine, initialize git:
   - `git init`
4. Add and commit all files:
   - `git add .`
   - `git commit -m "Initial commit"`
5. Add GitHub repo as remote:
   - `git remote add origin https://github.com/username/repo.git`
6. Push your local commits to GitHub:
   - `git push -u origin master` (or `main` depending on branch name).

cmd+shift+p or **>** in dialog of visual studio code opens the command pallete. **Shell Command: Install 'code' command in PATH** here should give terminal access to open visual studio code text editor on files in present working directory with **code .** in pwd.

next.js (laptop resolution 1920 – 890-930)

dialog in next.js the tsx code for a website is written in a file called **page.tsx**



each folder inside app has a page.tsx file which can be accessed by /foldername

layout.tsx file wraps all the page.tsx files in a project

to route between page we use next/link

import Link from 'next/link'; above & outside all tsx funtions
and then we can use the <Link> component.

```
<Link href="/red">
  <Button className="hidden md:inline-flex bg-transparent text-white/80 hover:text-white hover:bg-[#6c63fe] text-[1.77vh]">
    Light
  </Button>
</Link>
```

there is also router.push(/foldername) to route based on a action like onclick
(router here is a variable assigned by useRouter hook)

clerk

to run clerk on your nextjs app

```
npm  yarn  pnpm  bun

terminal

1  npm install @clerk/nextjs
```

add a middleware.ts file to your root (if no src) or in src directory

```
1   import { clerkMiddleware } from '@clerk/nextjs/server';
2
3   export default clerkMiddleware();
4
5   export const config = {
6     matcher: [
7       // Skip Next.js internals and all static files, unless found in search params
8       '/((?!_next|[^?]*\\.(?:html?|css|js(?!on)|jpe?g|webp|png|gif|svg|ttf|woff2?|ico|csv|docx?|xlsx?|zip|webmanifest)).*)',
9       // Always run for API routes
10      '/(api|trpc)(.*)',
11    ],
12  };
```

First we wrap <ClerkProvider> component in layout.tsx for the funtions to work.

```
import {
  ClerkProvider,
} from '@clerk/nextjs'
```

```
export default function RootLayout({
    children,
  }: {
    children: React.ReactNode
  }) {
    return (
      <ClerkProvider>
      <html lang="en">
        <head>
    <script src="https://cdn.jsdelivr.net/particles.js/2.0.0/particles.min.js"></script>
  </head>
        <body className={`${inter.className} min-h-screen bg-white text-black`}>
          {children}
        </body>
      </html>
      </ClerkProvider>
    )
  }
```

```
import {
  SignInButton,
  SignUpButton,
  SignedIn,
```

SignedOut,
UserButton,
} from '@clerk/nextjs'; for landing page use

```
<SignedOut>
  <SignInButton>
    <Button className="only-1366 bg-transparent text-white/80 hover:text-white hover:bg-[#6c63fe] md:text-[1.77vh] text-[4vw]">
      Login
    </Button>
  </SignInButton>
  <SignUpButton>
    <Button className="only-1366 bg-white text-gray-900 md:w-[5.75vw] w-[25vw] hover:bg-[#6c63fe] hover:text-white md:text-[1.77vh] text-[4vw]">
      Sign Up
    </Button>
  </SignUpButton>
</SignedOut>
```

<SignInButton> and <SignUpButton> are just components for Log-in & Sign-up

<SignedIn> and <SignedOut> components show different data based on if your signed in or not throughout the project

once logged in we can import { UserButton, useUser } from '@clerk/nextjs'; and then const { user, isLoaded } = useUser(); inside the main tsx function code to use the useUser hook to return an object “user” with all its information like user.username to be accessed.

```
<SignedOut>
  <RedirectToSignIn />
</SignedOut>
```

we can also import <RedirectToSignIn> component to redirect from inner page links if the user is signed out back to log-in page.

typescript

essentially just javascript but we can declare types for variables

```typescript
let variableName: type;
```

but if we don't declare any type it assumes type : any; which is flexible and also makes javascript code runnable on typescript compilers.

we can also declare our own datatypes like objects with type or interface

```typescript
type User = {
  name: string;
  age: number;
};
```

```typescript
interface User {
  name: string;
  age: number;
  greet(): void;
}
```

Feature	Type	Interface
Definition	Can define various data types including primitive, union, intersection, tuple, and object shapes.	Defines the shape of an object or a contract like properties and methods.
Flexibility	More flexible, supports union and intersection types.	Less flexible, mainly for object shapes, but supports declaration merging and extension.
Declaration merging	Not supported. Declaring the same type twice causes errors.	Supports declaration merging, allowing interfaces with the same name to combine.
Extending	Types can be combined using intersections (&).	Interfaces can extend other interfaces or types using `extends`.
Usage with primitives	Supports primitives, unions, tuples, etc.	Cannot describe primitive types directly.
Error messages and tooling	Sometimes less clear error messages with complex constructs.	Generally better error messages and more IDE-friendly.
Implementation in classes	Can't be implemented or extended by classes directly.	Can be implemented or extended by classes, good for OOP.

react hooks

react hooks need to be imported from react to be used

```
import { useRef, useState } from 'react';
```

useState

```jsx
const [stateVariable, setStateVariable] = useState(initialValue);
```

we can give a variable an initial value and a setter function so whenever setterfunction(new value) the variable will be set to new value and code will run based on new value as this variable will be stored in memory of the webpage.

useRouter

- Usage:

```jsx
const router = useRouter();
```

  - Access methods like `router.push('/path')`, `router.replace('/path')`, and others.

useRef

```js
const triggerRef = useRef<HTMLDivElement>(null);
```

triggerRef is assigned null for now but will eventually point to a div element

```js
<div ref={triggerRef} className="relative">
  <UserButton ... />
</div>
```

in this example we are assigning this specific div container as triggerRef which has a button inside of it.

```js
<div
  className="inline-flex ..."
  onClick={() => {
    const btn = triggerRef.current?.querySelector('button');
    btn?.click();
  }}
>
  <span>{user.username}</span>
  <div ref={triggerRef} className="relative">
    <UserButton ... />
  </div>
</div>
```

The outer div container wrapping the triggerRef div container has a onclick function that uses triggerRef.current? (which returns a mutable object that the triggerRef was pointing to) and use query selector to select the first element of type button inside the div container and execute it.

we basically make the outer a container a button that triggers the same thing as inner button would trigger which in this case is clerk userbutton.

```
<div ref={triggerRef} className="relative">
  <UserButton
    afterSignOutUrl="/"
    appearance={{
      elements: {
        userButtonPopoverCard: {
          transform: 'translateY(3.5vh)',
          '@media (max-width: 768px)': {
            transform: 'translateY(3.5vh) translateX(4vw)'
          }
        }
      }
    }}
  />
```

In clerk we can also adjust the location of userButtonPopoverCard using css translates.

react effect (already covered before but same concept)

react effects helps us synchronize our code when state values are changed.

lets say we want to tell the time a user has stayed on the site in seconds

running a setInterval inbuilt function in javascript that runs every 1000 ms

if we just use state to manage the seconds, every time the webpage rerenders the, a new setInterval is started, giving us multiple intervals running at the same time and each incrementing the count of seconds passed. useEffect ensured the interval only runs once as it runs after the rendering.

```
import { useEffect, useState } from "react";

export default function Clock() {
  const [counter, setCounter] = useState(0);

  useEffect(() => {
    const key = setInterval(() => {
      setCounter(count => count + 1)
    }, 1000);

    return () => {
      clearInterval(key);
    };
  }, [])

  return (
    <p>{counter} seconds have passed.</p>
  );
}
```

we have a clearInterval cleanup function because in development of react applications, the strict mode renders each component two times causing two intervals that can be avoided by cleaning it during unmounting (mounting is creation of a react component in the DOM and unmounting is removing it from the DOM). **arrow functions** => are used in stuff like count = count + 1 or functions being called because of async nature of javascript to ensure that these steps are done in a specific order and count is not incrementing some older previous value and resetting the count. useEffect must have a dependency matrix [] in itself which can be empty but if it has some variable or prop inside it, the interval will be set again when this variable or prop is changed

```
useEffect(
  () => {
    // execute side effect
    return () => {
      // cleanup function on unmounting or re-running effect
    }
  },
  // optional dependency array
  [/* 0 or more entries */]
)
```

useCallback

```js
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]); // will only change if a or b change
```

useEffect runs once on first render and run again everytime the dependency changes whereas useCallback gives us a function that can be run anytime by us but will be run again everytime dependency changes however the big difference is that it won't be run again on a new render (calling a function or component) it will keep its values until and only the dependencies change.

api routes

api routes are folders inside the app folder providing a backend without needing a separate server making each api route a endpoint automatically.

to send something to your api route (for any purpose like either saving in a database, sending information to a third party API for process or doing server side functions on the information and getting the information back)

```
const handleSendMessage = async () => {
  const messageText = inputValue.trim();

  if (messageText) {
    const newMessage: UserMessage = {
      id: Date.now(),
      text: messageText,
      sender: 'user',
      timestamp: new Date()
    };

    setMessages((prev: UserMessage[]) => [...prev, newMessage]);
    setInputValue('');
    setIsLoading(true);

    try {
      const financialData = calculateFinancialSummary();

      const response = await fetch('/api/gemini-financial', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          message: messageText,
          language: currentLanguage,
          financialData: financialData,
          userId: user?.id
        })
      });

      const data = await response.json();
      setIsLoading(false);

      const botMessageId = Date.now() + 1;
      const botResponse: UserMessage = {
        id: botMessageId,
        text: '',
        sender: 'bot',
        timestamp: new Date()
      };
      setMessages((prev: UserMessage[]) => [...prev, botResponse]);
      typeWriterEffect(data.message, botMessageId);

    } catch (error) {
      setIsLoading(false);
      const errorResponse: UserMessage = {
        id: Date.now() + 1,
        text: currentLanguage === 'hi'
          ? "क्षमा करें, कनेक्शन में समस्या है। कृपया फिर से कोशिश करें।"
          : "Sorry, I'm having trouble connecting. Please try again.",
        sender: 'bot',
        timestamp: new Date()
      };
      setMessages((prev: UserMessage[]) => [...prev, errorResponse]);
    }
  }
};

const handleKeyPress = (e: KeyboardEvent<HTMLInputElement>) => {
  if (e.key === 'Enter' && !e.shiftKey) {
    e.preventDefault();
    handleSendMessage();
  }
};

const formatTime = (date: Date): string => {
  return date.toLocaleTimeString('en-IN', {
    hour: '2-digit',
    minute: '2-digit'
  });
};
```

in this example the function handleSendMessage is run whenever a message is sent in the chatbot of this code; **async** is important as it means javascript code can not go forward until the await part of the function is executed inside.

we trim the inputted message of trailing whitespaces, then create a new object called **newMessage** strictly for frontend display with all the message information, appending the newest message to an array with all the previous messages.

```javascript
const response = await fetch('/api/gemini-financial', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    message: messageText,
    language: currentLanguage,
    financialData: financialData,
    userId: user?.id
  })
});
```

this is the most important part; we send the information via await fetch to the api route as method POST with headers to signifiy it's a json object and then the body for a completely newly designed json object to send now; the reply from api route is directly saved here only in the response variable

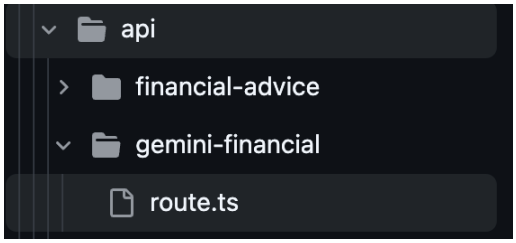
```
// TIME 3: Extract the data from the response
const data = await response.json();
// ↑ This RECEIVES the actual data from the response
```

and then the data variable extracts the http response as a json object.

the code in example then just shows some code for error handling; showing enter button only when keyboard is pressed & converting date to strings.

but the new variable data has the response & data.message now has the reply.

on the **api route** side inside the **route.ts** file in the specific api route folder




```
const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY!);

 const model = genAI.getGenerativeModel({
    model: 'gemini-1.5-flash',
    generationConfig: { temperature: 0.9 },
  });

const result = await model.generateContent(optimizedPrompt);
```

here the optimizedPrompt is just a variable we defined

```
const optimizedPrompt = `${systemPrompt}
```

and systemPrompt was the initial prompt variable shown on next page and since this is javascript code we can always use ${variablename} to always include our own variables inside the string of prompt. The variables that we sent in the JSON object from frontend

```
    const systemPrompt = language === 'hi'
      ? `आप BudgetBot हैं – एक दयालु, समझदार वित्तीय साथी जो उपयोगकर्ता की भावनाओं को समझते हैं।

वित्तीय स्थिति: ${financialContext}
${emotionalNote}

लक्ष्य:
– सहानुभूति और समझदारी दिखाना
– व्यावहारिक सलाह देना बिना जजमेंट के
– उम्मीद और प्रेरणा देना
– व्यक्तिगत स्पर्श जोड़ना

60–100 शब्दों में गर्मजोशी से जवाब दें।`

      : `You are BudgetBot – a compassionate, understanding financial companion who genuinely cares about user wellbeing.

Financial Status: ${financialContext}
${emotionalNote}

Mission:
– Show empathy and understanding
– Give practical advice without judgment
– Inspire hope and confidence
– Add personal touches
– Celebrate progress and acknowledge struggles

Respond in 60–100 words with warmth and care.`;

    const optimizedPrompt = `${systemPrompt}

User: "${message}"

Respond with genuine care and helpful guidance:`;
```

now this whole thing is obviously inside a specific async function

```
export async function POST(request: NextRequest) {
  let languageFallback: 'en' | 'hi' = 'en';

  try {
    if (!process.env.GEMINI_API_KEY) {
      return NextResponse.json({ error: 'Gemini API key not configured' }, { status: 500 });
    }

    const body: RequestBody = await request.json();
    const { message, language, financialData, userId } = body;
    languageFallback = language;

    if (!message) {
      return NextResponse.json({ error: 'Message is required' }, { status: 400 });
    }
```

here this function autoruns when a POST request is sent to the api route and inside it the body variable of type RequestBody (which is declared here as a object type above in the code) gets the request from frontend and is immediately stored four variables for use in the code as shown & NextRequest and NextResponse is just helper classes / object types imported at top.
import { NextRequest, NextResponse } from 'next/server';

and finally to return the information back to frontend

after the const result = await model.generateContent(optimizedPrompt); shown initially we put the result.response in new variable and textify it in text variable

```
  const response = await result.response;
  let text = response.text();
```

```ts
return NextResponse.json({
  message: text,
  source: 'gemini_fresh',
  cached: false,
  timestamp: new Date().toISOString(),
});
```

whatever we return in this POST function will be given back to frontend whether its this functional reply or some error handling return.

```
    if (!process.env.GEMINI_API_KEY) {
      return NextResponse.json({ error: 'Gemini API key not configured' }, { status: 500 });
    }
```

```
if (!message) {
  return NextResponse.json({ error: 'Message is required' }, { status: 400 });
}
```

connecting frontend to flask python backend

in our example

const fileInputRef = useRef<HTMLInputElement | null>(null);

this useRef hook is declared in the main function component topline

now later in the code we have a function triggered by a button click

```
 const handleButtonClick = () => {
  fileInputRef.current?.click();
 };
```

triggering the fileinputref html input element assigned to the input element

```
<input
        type="file"
        accept="image/*"
        ref={fileInputRef}
        onChange={handleFileChange}
        className="hidden"
      />
```

so after we input the image in type=file we trigger handleFileChange function

```
const handleFileChange = async (e: React.ChangeEvent<HTMLInputElement>) => {
  if (!e.target.files?.length) return;

  const file = e.target.files[0];
  setIsLoading(true);
  setUploadStatus('uploading');
  setError(null);

  const formData = new FormData();
  formData.append("file", file);

  try {
    const res = await fetch("http://127.0.0.1:5000/upload", {
      method: "POST",
      body: formData,
    });

    const uploadResult: UploadResult = await res.json();

    if (uploadResult.success) {
      setUploadStatus('success');
      localStorage.setItem('bioLuminescenceResult', JSON.stringify(uploadResult));

      // Save to dashboard history
      saveToHistory(uploadResult, file);

      // Show success animation for 1.5 seconds, then navigate
      setTimeout(() => {
        router.push('/result');
      }, 1500);
    } else {
      setUploadStatus('error');
      setError("Failed to process image");
      setIsLoading(false);
    }
  } catch (err) {
    console.error("Upload failed:", err);
    setUploadStatus('error');
    setError("Upload failed. Make sure the backend server is running.");
    setIsLoading(false);
  }
};
```

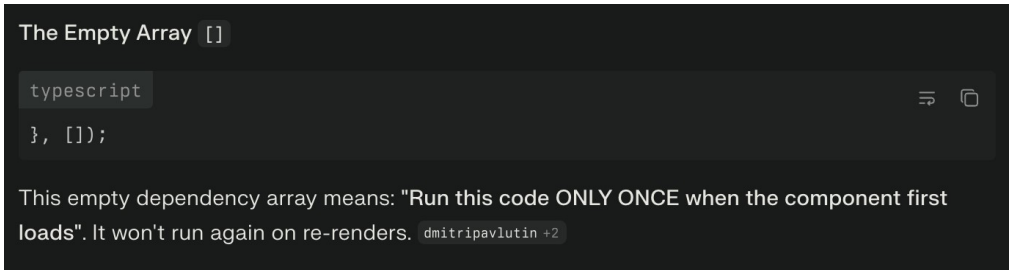
which is not that complicated; ultimately e is just a event object parameter which holds all information about the event that just occurred which in this case is a input element being triggered. e.target.files is just all files in e.target which is the element causing the event (in this case input element). we store the file in variable file and then create a new variable of type FormData which is just a javascript class format to send HTTP requests

```typescript
const formData = new FormData();
formData.append("file", file);
```

And then we just await fetch the result in a new variable called res and jsonify it in a new variable uploadResult which is then stored in local storage of browser as bioLuminescenceResult for access in other webpages.

then in another webpage to showcase python app reply

```
useEffect(() => {
  try {
    const stored = localStorage.getItem('bioLuminescenceResult');
    if (stored) {
      const parsed: UploadResult = JSON.parse(stored);
      if (parsed && parsed.success) {
        setResult(parsed);
        console.log("Retrieved result:", parsed);
      } else {
        setError('No valid result data found');
      }
    } else {
      setError('No analysis result found. Please upload an image first.');
    }
  } catch (e) {
    console.error('Error loading result:', e);
    setError('Failed to load analysis result');
  } finally {
    setIsLoading(false);
  }
}, []);
```



The bioLuminescenceResult is put in variable stored and then JSON.parse converts a JSON into a object that can be accessed & used in this case in the variable parsed of type UploadResult. If parsed is a success then setResult function is set to rendered again due to the useState declared on top

const [result, setResult] = useState<UploadResult | null>(null);

as you can see this line at the top of main function component is triggered since result is no longer null and is parsed variable of type UploadResult.

once result is passed and component re renders; all places accessing result will be able to show its class members like result.image or result.unique_species.

In the app.py part

```python
from flask import Flask, request, jsonify, send_file
from flask_cors import CORS
app = Flask(__name__)
CORS(app)
```

app=Flask(__name__) creates a webserver and CORS(app) enables cross origin resource sharing that allows communication between different ports on network

```python
@app.route('/upload', methods=['POST'])
def upload_file():
    try:
        if 'file' not in request.files:
            return jsonify({'error': 'No file provided'}), 400

        file = request.files['file']
        if file.filename == '':
            return jsonify({'error': 'No file selected'}), 400

        # Generate unique filename
        file_id = str(uuid.uuid4())
        filename = f"{file_id}.jpg"
        filepath = os.path.join(UPLOAD_FOLDER, filename)

        # Save uploaded file
        file.save(filepath)
```

@app.route part is a decorator that ensures that whenever someone sends a POST request to this port on the network; run the upload_file() function

file is the key in the object of key value pairs sent by the javascript frontend

we check for errors with file's presence & then we save its members in variables

and then run the model and save the results in variables that can be sent back as the return section of this same upload_file() function

```python
        return jsonify({
            'success': True,
            'image': f"{img_base64}",
            'detections': detections,
            'total_count': total_count,
            'unique_species': unique_species,
            'species_summary': species_summary,
            'file_id': file_id
        })
```

where jsonify converts python dictionaries into json objects while the image itself is being sent back as base64 data of the jpeg data of the image for safe text communication of images.

```python
if __name__ == '__main__':
    app.run(debug=True, host='127.0.0.1', port=5000)
```

and the final lines for running our named flask server on whatever port of link.

CRUD

in large works the same way of sending data to api routes and then we can save there; after installing supabase in project we can import it via

import supabase from '@/lib/supabase';

then we can just await json object

const { text, type } = await req.json();

and then use this syntax

```
const { data, error } = await supabase
  .from('user_inputs')
  .insert({
    user_id: userId,
    prompt_type: type,
    content: text,
  })
  .select(); // Add .select() to return the inserted data
```

here .from('user_inputs') line just denotes table kinda like INSERT INTO

then we .insert our data and .select() to send it back to us
as a way to verify what data is saved in the database and then return success

return NextResponse.json({ success: true, data });

to get this information back from database we need a **GET request**

```
import { NextResponse } from 'next/server';
import { auth } from '@clerk/nextjs/server';
import supabase from '@/lib/supabase';

export async function GET() {
  try {
    const { userId } = await auth();

    if (!userId) {
      return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
    }

    // Get the latest goal and code for this user
    const { data: goalData, error: goalError } = await supabase
      .from('user_inputs')
      .select('content, created_at')
      .eq('user_id', userId)
      .eq('prompt_type', 'goal')
      .order('created_at', { ascending: false })
      .limit(1);
```

```
    const { data: codeData, error: codeError } = await supabase
      .from('user_inputs')
      .select('content, created_at')
      .eq('user_id', userId)
      .eq('prompt_type', 'code')
      .order('created_at', { ascending: false })
      .limit(1);

    if (goalError || codeError) {
      console.error('Supabase error:', goalError || codeError);
      return NextResponse.json({ error: 'Database error' }, { status: 500 });
    }

    const latestGoal = goalData?.[0]?.content || null;
    const latestCode = codeData?.[0]?.content || null;

    return NextResponse.json({
      goal: latestGoal,
      code: latestCode,
      hasData: !!(latestGoal && latestCode)
    });

  } catch (err) {
    console.error('API error:', err);
    return NextResponse.json({ error: 'Internal server error' }, { status: 500 });
  }
}
```

which just mimics a SQL QUERY & calling this GET API function will send back the information back to frontend where we can do anything this this permanently stored data.

If you are wondering when do we process.env the supabase keys in this api route, its done in the lib utility section we are importing here into the supabase variable

```
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = process.env.SUPABASE_URL!
const supabaseAnonKey = process.env.SUPABASE_ANON_KEY!

const supabase = createClient(supabaseUrl, supabaseAnonKey)

export default supabase
```

payment gateway using RazorPay

very easy to add components with their two API keys

stock market api calls using finnhub

60 requests a minute

Popular Jaw-Dropping API Categories

- **AI & Machine Learning** (e.g., OpenAI, Gemini, Hugging Face): Add image captioning, text summarization, chatbot, voice assistants, or even real-time translation features.
- **Authentication & Authorization** (e.g., Auth0, Supabase Auth): Seamless sign-up/login with social providers, magic links, or OTPs.
- **Maps & Geolocation** (e.g., Google Maps, Mapbox, IP Geolocation): Interactive maps, place suggestions, travel routes, or location-based notifications.
- **Weather & Environment** (e.g., OpenWeatherMap, AirVisual): Personalized weather alerts, air quality monitoring, or climate dashboards.
- **Media & Entertainment** (e.g., Spotify, The Movie DB, YouTube Data): Show trending music/movies, enrich user profiles with favorite tracks, or pull memes/videos dynamically.
- **Messaging & Notifications** (e.g., Twilio, SendGrid, Pusher): Enable SMS or WhatsApp alerts, or real-time app notifications.
- **Finance/Payments** (e.g., Stripe, Plaid): Allow payments, budgeting, or bank account linking.
- **Health/Fitness** (e.g., Nutritionix, Fitbit): Calorie counters, meal suggestions, activity tracking.
- **Fun/Trivia APIs** (e.g., Numbers API, JokeAPI): Random trivia, jokes, or facts on dashboards.

Advanced or Unusual APIs

- **OCR/Image Recognition** (e.g., Google Vision, Azure OCR): Scan receipts, extract data from uploaded images.
- **Blockchain/Web3** (e.g., Moralis, Alchemy): NFT minting, crypto wallet connect, decentralized logins.

- **Speech/Voice AI** (e.g., Deepgram, AssemblyAI): Live transcription, voice commands, voice-to-text chatbots.
- **Gamification** (e.g., BadgeOS, Open Trivia): Award badges for milestones, create multiplayer quizzes.
- **Emotion & Sentiment Analysis** (e.g., Twinword, ParallelDots): Detect mood from text/voice, create mood-based interfaces.
- **Augmented Reality** (e.g., 8th Wall, Zappar): Browser-based AR experiences.
- **Public Data APIs** (e.g., NASA, Museums, Wikipedia): Fetch images from Mars, display random facts, create educational visualizations.

tailwind quick reference

```tsx
text-xs      // 0.75rem / 12px
text-sm      // 0.875rem / 14px
text-base    // 1rem / 16px (default)
text-lg      // 1.125rem / 18px
text-xl      // 1.25rem / 20px
text-2xl     // 1.5rem / 24px
text-3xl     // 1.875rem / 30px
text-4xl     // 2.25rem / 36px
text-5xl     // 3rem / 48px
text-6xl     // 3.75rem / 60px
```

```tsx
justify-start      // align items at start
justify-center     // center items
justify-end        // align items at end
justify-between    // equal space between items
justify-around     // equal space around items
justify-evenly     // equal space evenly
```

align-content (Cross axis spacing in multi-line flex/grid)

```tsx
content-start
content-center
content-end
content-between
content-around
content-evenly
```

align-items (Cross axis alignment in flex/grid containers)

```tsx
items-start
items-center
items-end
items-baseline
items-stretch
```

Flex and Flex Direction

```tsx
flex              // displays element as flex container (row direction default)
flex-row          // flex direction row (default)
flex-col          // flex direction column
flex-wrap         // wrap items to next line
flex-nowrap       // prevent wrapping
flex-grow         // allow item to grow to fill space
flex-shrink       // allow item to shrink
flex-auto         // grow and shrink as needed
```

Margin (m) and Padding (p) Shorthand

- `m` = margin, `p` = padding
- Directions:
  - `t` = top
  - `r` = right
  - `b` = bottom
  - `l` = left
  - `x` = left and right
  - `y` = top and bottom

Examples:

```tsx
mt-4   // margin-top: 1rem (16px)
mr-2   // margin-right: 0.5rem (8px)
mb-6   // margin-bottom: 1.5rem (24px)
ml-1   // margin-left: 0.25rem (4px)
mx-3   // margin-left & right: 0.75rem (12px)
my-5   // margin-top & bottom: 1.25rem (20px)

pt-2   // padding-top: 0.5rem (8px)
pr-6   // padding-right: 1.5rem (24px)
pb-1   // padding-bottom: 0.25rem (4px)
pl-4   // padding-left: 1rem (16px)
px-5   // padding-left & right: 1.25rem (20px)
py-2   // padding-top & bottom: 0.5rem (8px)
```

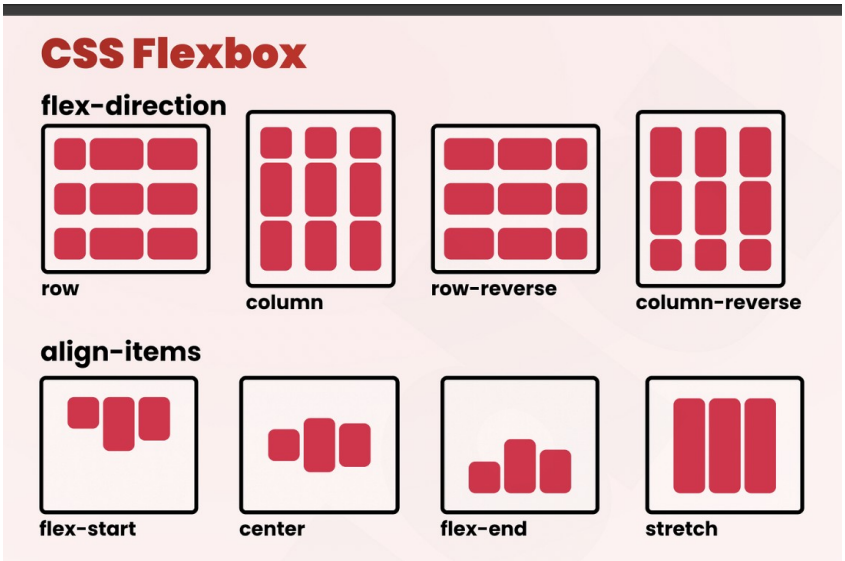
Font Weights

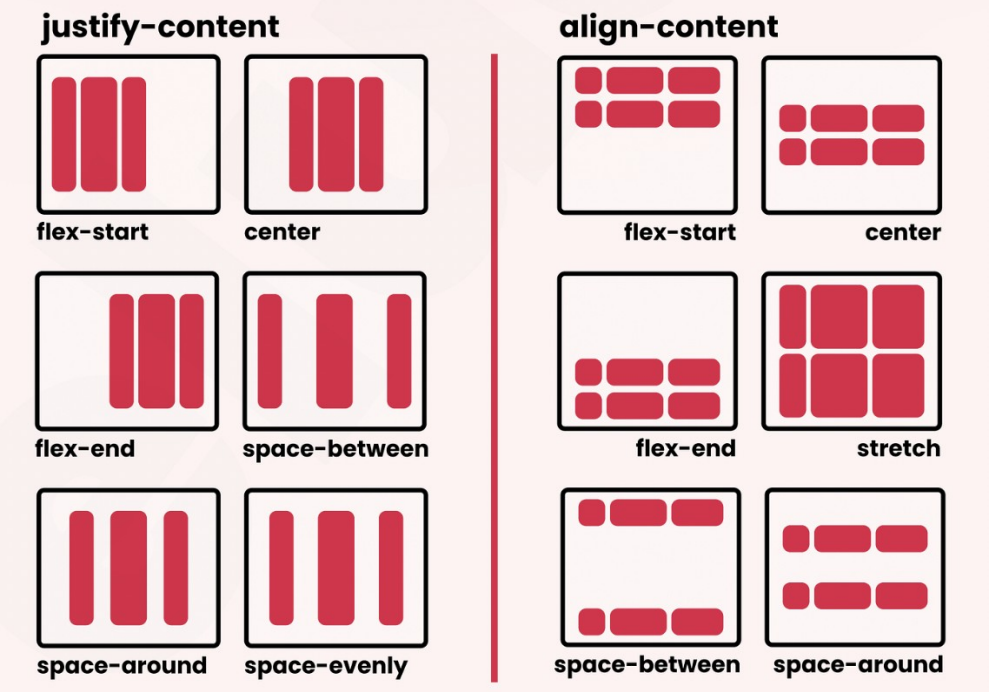
```tsx
font-thin       // 100
font-extralight // 200
font-light      // 300
font-normal     // 400 (default)
font-medium     // 500
font-semibold   // 600
font-bold       // 700
font-extrabold  // 800
font-black      // 900
```

```tsx
<div className="flex flex-col items-center justify-between p-4 text-lg
font-semibold">
  Hello Tailwind!
</div>
```

md : and normal tailwind styling ->

```tsx
<div className="text-sm md:text-lg">
  This text is small on mobile/smaller screens,
  but large (lg) on medium screens (≥768px) and larger.
</div>

<div className="bg-red-500 md:bg-blue-500">
  This background is red on smaller screens,
  but changes to blue on medium screens and above.
</div>
```





double code for responsive design

```tsx
"use client";

import { useRouter } from "next/navigation";
import { useEffect, useState } from "react";

// ------------------ Desktop Component --------------------

function HomePageDesktop() {
  const router = useRouter();

  useEffect(() => {
    // Initialize desktop-specific scripts, e.g., particles.js
    // ...
  }, []);

  return (
    <>
      {/* Desktop Header */}
      <header className="hidden md:flex bg-gray-800 text-white p-4">
        <h1 className="text-xl font-bold">Desktop Header</h1>
        {/* Desktop nav items */}
        {/* ... */}
      </header>

      {/* Desktop Main Section */}
      <main className="hidden md:block p-6">
        <h2 className="text-3xl">Welcome Desktop User</h2>
        {/* Desktop specific content */}
        {/* ... */}
      </main>
    </>
  );
}
```

```
function HomePageMobile() {
  const router = useRouter();

  useEffect(() => {
    // Initialize mobile-specific scripts
    // ...
  }, []);

  return (
    <>
      {/* Mobile Header */}
      <header className="flex md:hidden bg-gray-900 text-white p-3">
        <h1 className="text-lg font-semibold">Mobile Header</h1>
        {/* Mobile nav or hamburger menu */}
        {/* ... */}
      </header>

      {/* Mobile Main Section */}
      <main className="block md:hidden p-4">
        <h2 className="text-xl">Welcome Mobile User</h2>
        {/* Mobile specific content */}
        {/* ... */}
      </main>
    </>
  );
}

// ------------------ Main Responsive Wrapper --------------------

export default function HomePage() {
  const [isMobile, setIsMobile] = useState(false);

  useEffect(() => {
    const checkDevice = () => {
      setIsMobile(window.innerWidth < 768); // md breakpoint
    };

    checkDevice();
    window.addEventListener("resize", checkDevice);
    return () => window.removeEventListener("resize", checkDevice);
  }, []);

  return isMobile ? <HomePageMobile /> : <HomePageDesktop />;
}
```