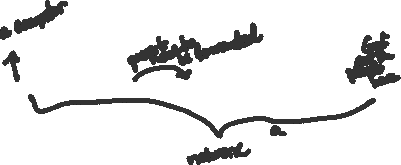
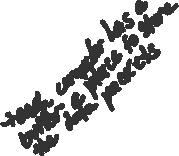
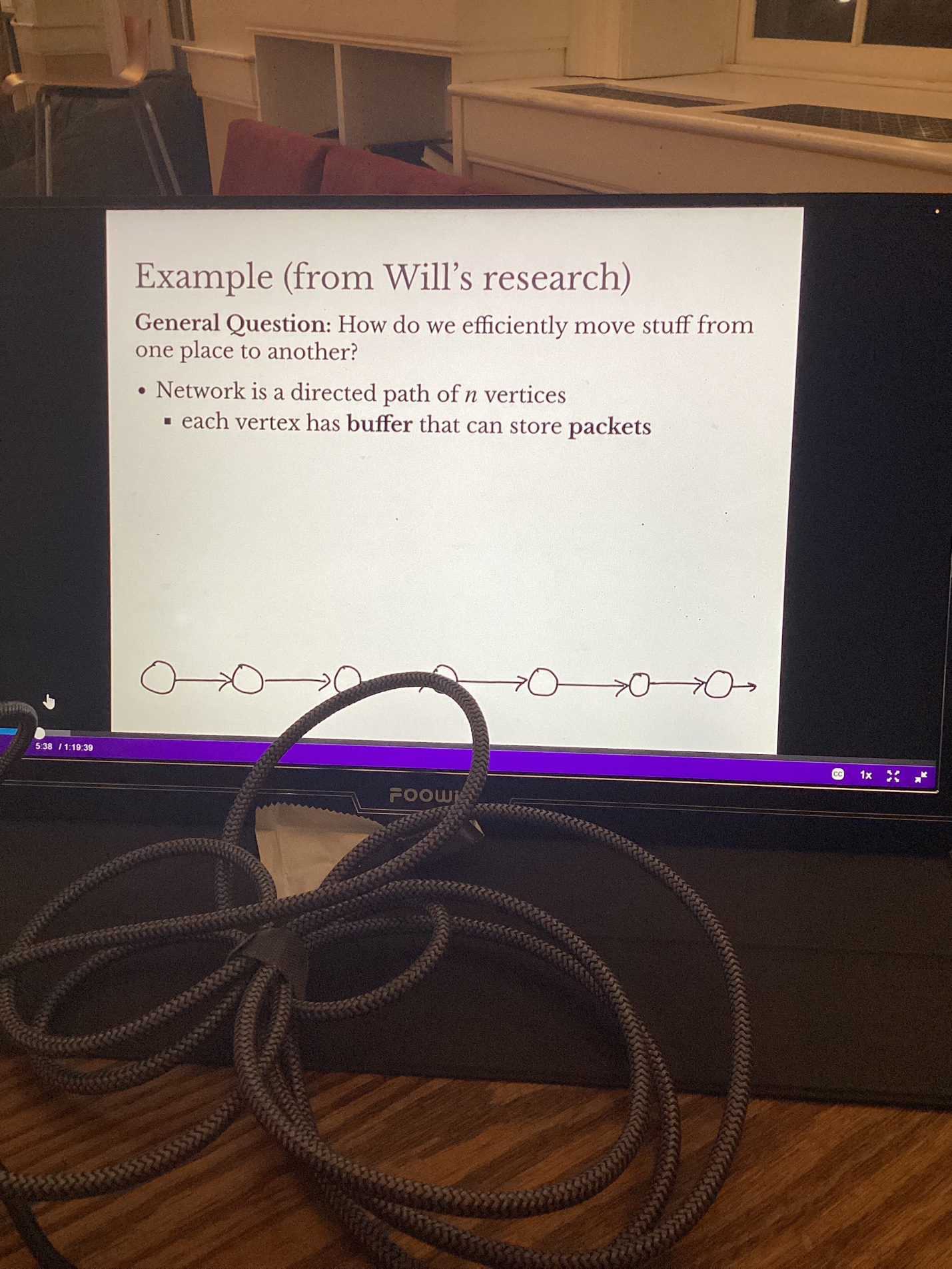
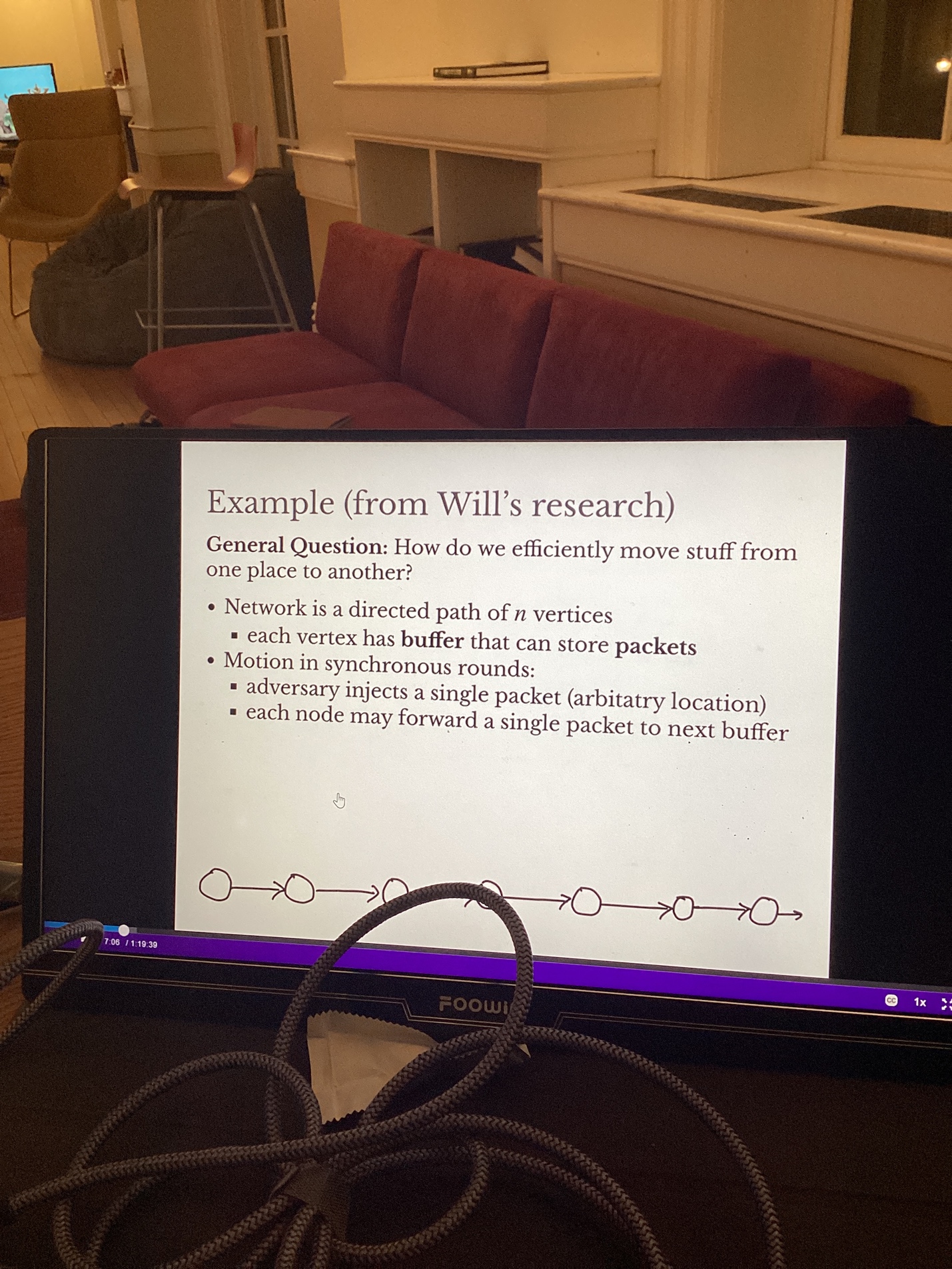
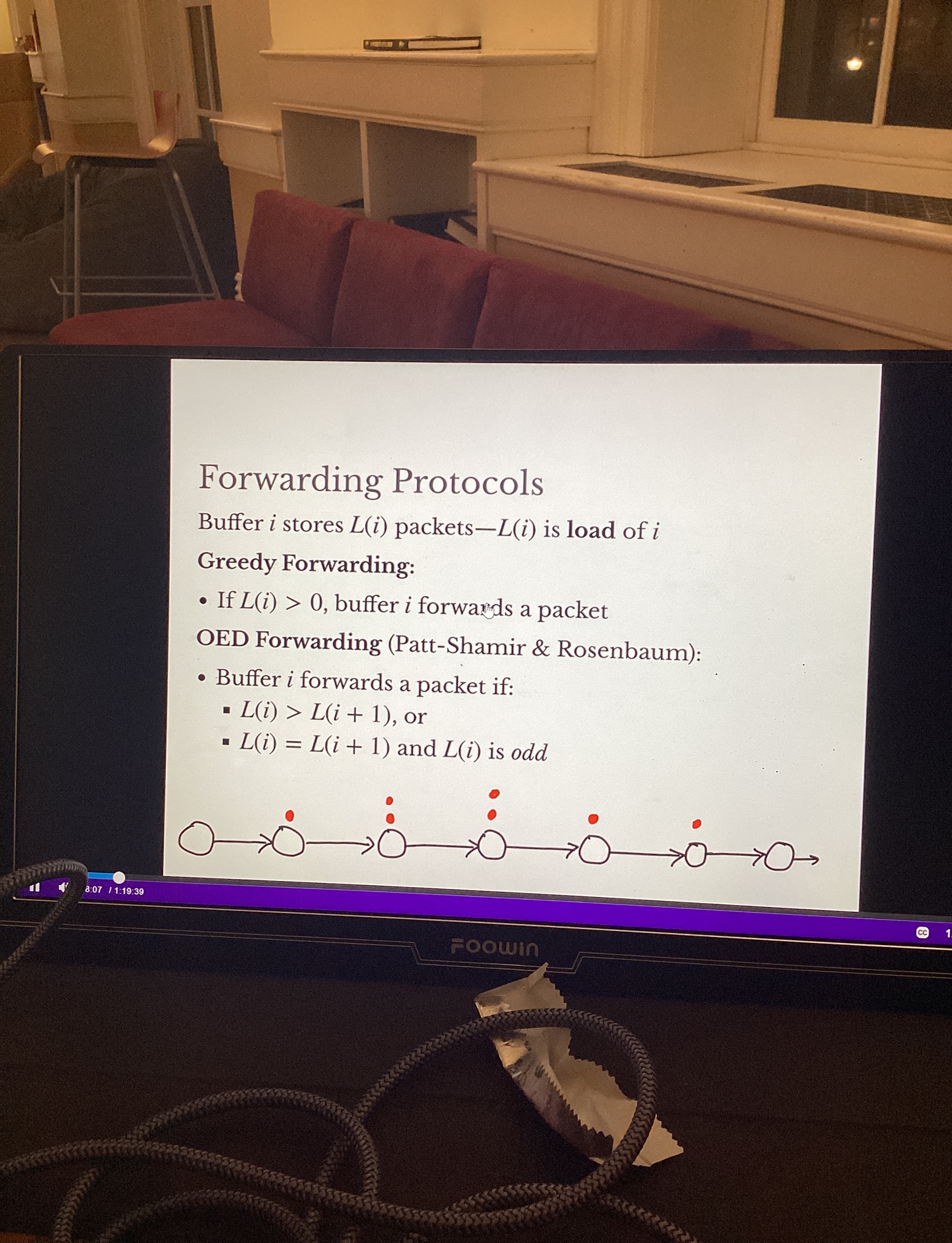
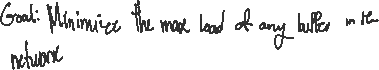
# Week 1

How to efficiently move stuff from one place to another? Stuff = data packets







# Algorithms

1. Identify the problem that you want to improve. What is the simplest thing that can improve it?

Example: you don’t want vertical package build up. So how can we make it distribute it horizontally instead?

Create animation for algorithms because they are sharable and you can see what the algorithm better

How can we make algorithms that make visualizations look nice? How can I write algorithms that look nice?

Think about concepts as opposed to remembering syntax. Example: how to build thinks from the ground up.

# Browser Inspection Tools

Firefox is best

Gives source code for a site

Hovering over the code/part of the website will show you what it corresponds to

The margin/paddings will correspond to whatever you are hovering over

You can edit code in the web dev tools, but it won’t change the code file

Where it says “element {}” you can change the CSS for the element you are hovering over

# HTML

HTML is a tree. Draw a tree to understand hierarchal relationships

Text

Description automatically generated

Doesn’t specify how the content looks, just what the content is. Its not a programming language

If you don’t specify how things should be displayed, for example, the font with CSS, the web browser will decide for you

[HTML](https://developer.mozilla.org/en-US/docs/Glossary/HTML) (HyperText Markup Language) is a markup language that tells web browsers how to structure the web pages you visit.

<!DOCTYPE html> It's a declaration that tells the browser which version of HTML we are using. It is not an HTML tag, it is an instruction to the web browser about what version of HTML the page is written in.

<html></html>: The <html> element. This element wraps all the content on the page. It is sometimes known as the root element.

<head></head>: The <head> element. This element acts as a container for everything you want to include on the HTML page, that isn't the content the page will show to viewers. This includes keywords and a page description that would appear in search results, CSS to style content, character set declarations links to custom favicons, and other metadata (data about the HTML, such as the author, and important keywords that describe the document). Web browsers use information contained in the [head](https://developer.mozilla.org/en-US/docs/Glossary/Head) to render the HTML document correctly.

<meta charset="utf-8">: The <meta> element. This element represents metadata that cannot be represented by other HTML meta-related elements, like <base>, <link>, <script>, <style> or <title>. The charset attributes sets the character set for your document to UTF-8, which includes most characters from the vast majority of human written languages. With this setting, the page can now handle any textual content it might contain. There is no reason not to set this, and it can help avoid some problems later.

* The [<title>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/title) element is metadata that represents the title of the overall HTML document (not the document's content.). if you try bookmarking the page (*Bookmarks > Bookmark This Page* or the star icon in the URL bar in Firefox), you will see the <title> contents filled in as the suggested bookmark name

Many <meta> elements include name and content attributes:

* name specifies the type of meta element it is; what type of information it contains.
* content specifies the actual meta content.

Two such meta elements that are useful to include on your page define the author of the page, and provide a concise description of the page

Graphical user interface, text

Description automatically generated

**Note:** Many <meta> features just aren't used anymore. For example, the keyword <meta> element (<meta name="keywords" content="fill, in, your, keywords, here">) — which is supposed to provide keywords for search engines to determine relevance of that page for different search terms — is ignored by search engines, because spammers were just filling the keyword list with hundreds of keywords, biasing results.

A favicon can be added to your page by:

1. Saving it in the same directory as the site's index page, saved in .ico format (most also support favicons in more common formats like .gif or .png)
2. Adding the following line into your HTML's [<head>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/head) block to reference it:

A picture containing diagram

Description automatically generated

The [<link>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link) element should always go inside the head of your document. This takes two attributes, rel="stylesheet", which indicates that it is the document's stylesheet, and href, which contains the path to the stylesheet file



The [<script>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script) element should also go into the head, and should include a src attribute containing the path to the JavaScript you want to load, and defer, which basically instructs the browser to load the JavaScript after the page has finished parsing the HTML. This is useful as it makes sure that the HTML is all loaded before the JavaScript runs, so that you don't get errors resulting from JavaScript trying to access an HTML element that doesn't exist on the page yet.



You can set the language of your page. This can be done by adding the [lang attribute](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/lang) to the opening HTML ta. You can also set subsections of your document to be recognized as different languages

A picture containing shape

Description automatically generated

<body></body>: The [<body>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/body) element. This contains *all* the content that displays on the page, including text, images, videos, games, playable audio tracks, or whatever else

No matter how much whitespace you use inside HTML element content (which can include one or more space character, but also line breaks), the HTML parser reduces each sequence of whitespace to a single space when rendering the code

# Elements

## <em> </em> italics

Syntax is that it shows in italics but the semantics is that it “emphasizes”

## <strong> </strong> bold

<p> </p> paragraph 🡪 text will stand on its own

<a> </a> link

Tags in HTML are not case-sensitive. This means they can be written in uppercase or lowercase. For example, a [<title>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/title) tag could be written as <title>, <TITLE>, <Title>, <TiTlE>, etc., and it will work. However, it is best practice to write all tags in lowercase for consistency and readability

## Void Elements

Some elements consist of a single tag, which is typically used to insert/embed something in the document. Such elements are called [void elements](https://developer.mozilla.org/en-US/docs/Glossary/Void_element). There is no requirement to add a / at the end of a void element's tag However, it is also a valid syntax, and you may do this when you want your HTML to be valid XML

Self closing tags

Text doesn’t go inside of it so theres no need to have an opening tag.

Example: <br/> breaks up a line

Web browsers will render incorrect code

# Block Level Elements

Block-level elements form a visible block on a page. A block-level element appears on a new line following the content that precedes it. Any content that follows a block-level element also appears on a new line. Block-level elements are usually structural elements on the page. For example, a block-level element might represent headings, paragraphs, lists, navigation menus, or footers. A block-level element wouldn't be nested inside an inline element, but it might be nested inside another block-level element

Nested elements give a three-like structure. The outer element is the parent of the inner element

Text

Description automatically generated

# Inline Elements

Inline elements are contained within block-level elements, and surround only small parts of the document's content (not entire paragraphs or groupings of content). An inline element will not cause a new line to appear in the document. It is typically used with text, for example an [<a>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a) element creates a hyperlink, and elements such as [<em>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/em) or [<strong>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/strong) create emphasis

Note: The terms block and inline, as used in this article, should not be confused with [the types of CSS boxes](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model#block_and_inline_boxes) that have the same names. While the names correlate by default, changing the CSS display type doesn't change the category of the element, and doesn't affect which elements it can contain and which elements it can be contained in

## Block vs Inline

[<em>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/em) is an inline element. As you see below, the first three elements sit on the same line, with no space in between. On the other hand, [<p>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p) is a block-level element. Each p element appears on a new line, with space above and below. (The spacing is due to default [CSS styling](https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps) that the browser applies to paragraphs.)

Graphical user interface, text

Description automatically generated with medium confidenceTable

Description automatically generated

Text

Description automatically generated with medium confidence

Paragraph is a block element, it creates space above, below and takes up a line

<em></em> is an inline element. They don’t create a new horizontal line

## HTML5

**Both HTML and HTML5 are hypertext markup languages**, primarily used to develop web pages or applications. HTML5 is the latest version of HTML and supports new markup language functionalities such as multimedia, new tags and elements as well as new APIs. HTML5 also supports audio and video

# Attributes

Elements can have attributes. Attributes contain extra information about the element that won't appear in the content> Different elements have different attributes

Graphical user interface, website

Description automatically generated

An attribute should have:

* A space between it and the element name. (For an element with more than one attribute, the attributes should be separated by spaces too.)
* The attribute name, followed by an equal sign.
* An attribute value, wrapped with opening and closing quote marks

## Examples

href

This attribute's value specifies the web address for the link. For example: href="https://www.mozilla.org/".

title

The title attribute specifies extra information about the link, such as a description of the page that is being linked to.

target

The target attribute specifies the browsing context used to display the link. For example, target="\_blank" will display the link in a new tab. If you want to display the linked content in the current tab, just omit this attribute.

## [Boolean attributes](https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started#boolean_attributes)

Sometimes you will see attributes written without values. This is entirely acceptable. These are called Boolean attributes. Boolean attributes can only have one value, which is generally the same as the attribute name

 OR 

If you look at code for a lot of other sites, you might come across a number of strange markup styles, including attribute values without quotes. This is permitted in certain circumstances, but it can also break your markup in other circumstances. Always include the attribute quotes. It can be single or double quotes.

A picture containing text

Description automatically generated

Note: To use quote marks inside other quote marks of the same type (single quote or double quote), use [HTML entities](https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started" \l "entity_references_including_special_characters_in_html).





## Entity References

In HTML, the characters <, >,",' and & are special characters. They are parts of the HTML syntax itself. If you want to use an ampersand or less-than sign, and not have it interpreted as code, you do this with character references. These are special codes that represent characters, to be used in these exact circumstances. Each character reference starts with an ampersand (&), and ends with a semicolon (;)

Table

Description automatically generated

Comments

Browsers ignore comments, effectively making comments invisible to the user. To write an HTML comment, wrap it in the special markers <!-- and -->

# Text

[<h1>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<h2>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<h3>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<h4>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<h5>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), and [<h6>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements). Each element represents a different level of content in the document; <h1> represents the main heading, <h2> represents subheadings, <h3> represents sub-subheadings, and so on. It's really up to you what the elements involved represent, as long as the hierarchy makes sense

Best Practices

* Preferably, you should use a single <h1> per page—this is the top level heading, and all others sit below this in the hierarchy.
* Of the six heading levels available, you should aim to use no more than three per page, unless you feel it is necessary. Documents with many levels (for example, a deep heading hierarchy) become unwieldy and difficult to navigate. On such occasions, it is advisable to spread the content over multiple pages if possible.
* Search engines indexing your page consider the contents of headings as important keywords for influencing the page's search rankings. Without headings, your page will perform poorly in terms of [SEO](https://developer.mozilla.org/en-US/docs/Glossary/SEO) (Search Engine Optimization).

Lists

Unordered lists are used to mark up lists of items for which the order of the items doesn't matter.

A picture containing text

Description automatically generated

Ordered lists are lists in which the order of the items does matter.

Graphical user interface, text, application, email

Description automatically generated

It is perfectly OK to nest one list inside another one.

Text, letter

Description automatically generated

In HTML we use the [<em>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/em) (emphasis) element to mark up such instances.  Browsers style this as italic by default, but you shouldn't use this tag purely to get italic styling. To do that, you'd use a [<span>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span) element and some CSS, or perhaps an [<i>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/i) element

# [Multimedia and Embeddings](https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding)

## Images

### <img> element/tag

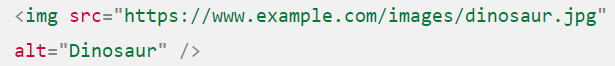
1. Void element 🡪 can not have any child content and can not have an end tag
2. Elements like [<img>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img) and [<video>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video) are sometimes referred to as **replaced elements**. This is because the element's content and size are defined by an external resource (like an image or video file), not by the contents of the element itself
3. Requires two attributes to be useful: src and alt
4. **src 🡪** The src attribute contains a URL pointing to the image you want to embed in the page. The src attribute can be a relative URL or an absolute URL. Without a src attribute, an img element has no image to load.

Relative



Absolute

This not recommended



You should host your own images, which in simple setups means keeping the images for your website on the same server as your HTML. In more advanced setups, you might use a [CDN (Content Delivery Network)](https://developer.mozilla.org/en-US/docs/Glossary/CDN) to deliver your images

Warning: **Never** point your src attribute at an image hosted on someone else's website **without permission**. This is called "hotlinking". It is generally considered unethical, since someone else would be paying the bandwidth costs for delivering the image when someone visits your page

1. **alt 🡪** Its value is supposed to be a textual description of the image, for use in situations where the image cannot be seen/displayed or takes a long time to render because of a slow internet connection
   1. use cases 🡪 The browser doesn't support the image type. Some people still use text-only browsers, such as [Lynx](https://en.wikipedia.org/wiki/Lynx_%28web_browser%29), which displays the alt text of images.

You may want to provide text for search engines to utilize; for example, search engines can match alt text with search queries.

Users have turned off images to reduce data transfer volume and distractions. This is especially common on mobile phones, and in countries where bandwidth is limited or expensive

1. You can use the width and height (not part of the style attribute) attributes to specify the width and height of your image based on the specs of the image. If the image hasn't yet loaded, you'll notice the browser is leaving a space for the image to appear in. **This is a good thing to do, resulting in the page loading quicker and more smoothly.** However, you shouldn't alter the size of your images using HTML attributes. If you set the image size too big, you'll end up with images that look grainy, fuzzy, or too small, and wasting bandwidth downloading an image that is not fitting the user's needs. You should use an image editor to put your image at the correct size before putting it on your webpage. You want to maintain the aspect ratio
2. You should use [CSS background images](https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Images_in_HTML#css_background_images) for decorative images
3. you can also add title attributes to images, to provide further supporting information if needed. This gives us a tooltip on mouse hover, just like link titles Graphical user interface, text

   Description automatically generated

Titles are not recommended. Its better to put the image description and supporting information in the main article text, rather than attached to the image

1. Captions: there would be nothing to stop you from doing this:

**Graphical user interface, text, application

Description automatically generated**

But there is a problem here: there is nothing that semantically links the image to its caption, which can cause problems for screen readers. For example, when you have 50 images and captions, which caption goes with which image?

A better solution, is to use the HTML [<figure>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/figure) and [<figcaption>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/figcaption) elements. These are created for exactly this purpose: to provide a semantic container for figures, and to clearly link the figure to the caption.

The [<figcaption>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/figcaption) element tells browsers, and assistive technology that the caption describes the other content of the [<figure>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/figure) element.

Captions benefit even people who can see the image, whereas [alt](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img#attr-alt) text provides the same functionality as an absent image. Therefore, captions and alt text shouldn't just say the same thing, because they both appear when the image is gone

A figure doesn't have to be an image. It is an independent unit of content that:

* Expresses your meaning in a compact, easy-to-grasp way.
* Could go in several places in the page's linear flow.
* Provides essential information supporting the main text.

A figure could be several images, a code snippet, audio, video, equations, a table, or something else.

he CSS [background-image](https://developer.mozilla.org/en-US/docs/Web/CSS/background-image) property, and the other background-\* properties, are used to control background image placement. For example, to place a background image on every paragraph on a page, you could do this

Logo, company name

Description automatically generated

So why bother with HTML images? As hinted to above, CSS background images are for decoration only

They can't have any text equivalents, are invisible to screen readers, and so on

Graphical user interface, text, application

Description automatically generated

# Adding Vector Graphics (SVG) to Web

Vector graphics is a form of computer graphics in which visual images are created directly from geometric shapes defined on a Cartesian plane, such as points, lines, curves and polygons. Vector graphics are **computer images created using a sequence of commands or mathematical statements that place lines and shapes in a two-dimensional**

Vector graphics are very useful in many circumstances — they have small file sizes and are highly scalable, so they don't pixelate when zoomed in or blown up to a large size

**Raster images** are defined using a grid of pixels — a raster image file contains information showing exactly where each pixel is to be placed, and exactly what color it should be. Popular web raster formats include Bitmap (.bmp), PNG (.png), JPEG (.jpg), and GIF (.gif.)

**Vector images** are defined using algorithms — a vector image file contains shape and path definitions that the computer can use to work out what the image should look like when rendered on the screen. The [SVG](https://developer.mozilla.org/en-US/docs/Glossary/SVG) format allows us to create powerful vector graphics for use on the Web.

Example

The difference is that the left one is a PNG, and the right one is an SVG image.

The difference becomes apparent when you zoom in the page — the PNG image becomes pixelated as you zoom in because it contains information on where each pixel should be (and what color). When it is zoomed, each pixel is increased in size to fill multiple pixels on screen, so the image starts to look blocky. The vector image however continues to look nice and crisp, because no matter what size it is, the algorithms are used to work out the shapes in the image, with the values being scaled as it gets bigger.

[A picture containing clipart, outdoor object

Description automatically generated](https://mdn.github.io/learning-area/html/multimedia-and-embedding/adding-vector-graphics-to-the-web/vector-versus-raster.html)

vector image files are much lighter than their raster equivalents, because they only need to hold a handful of algorithms, rather than information on every pixel in the image individually What is SVG?

SVG is an XML-based language for describing vector images. It's basically markup, like HTML, except that you've got many different elements for defining the shapes you want to appear in your image, and the effects you want to apply to those shapes. SVG is for marking up graphics, not content. At the simplest end of the spectrum, you've got elements for creating simple shapes, like <circle> and <rect>. More advanced SVG features include <feColorMatrix> (transform colors using a transformation matrix,) <animate> (animate parts of your vector graphic,) and <mask> (apply a mask over the top of your image.)

The version and baseProfile attributes are used to specify the version of SVG that is being used.

Xmlns 🡪 It is a namespace declaration. It declares the default namespace for the document. A namespace is an identifier that can be used to group elements and attributes. The purpose of namespaces is to avoid name collisions between elements and attributes in XML documents.

The argument passed in to xmlns is a URL that points to the XML Schema for SVG documents. The URL is "[http://www.w3.org/2000/svg"](http://www.w3.org/2000/svg%22). It tells the browser that the code is SVG and not HTML.

Text

Description automatically generated with medium confidence

For creating SVG images, most people use a vector graphics

editor like [Inkscape](https://inkscape.org/) or  [Illustrator](https://en.wikipedia.org/wiki/Adobe_Illustrator)

## SVG Benefits

* Text in vector images remains accessible (which also benefits your [SEO](https://developer.mozilla.org/en-US/docs/Glossary/SEO)).
* SVGs lend themselves well to styling/scripting, because each component of the image is an element that can be styled via CSS or scripted via JavaScript

## SVG does have some disadvantages:

* SVG can get complicated very quickly, meaning that file sizes can grow; complex SVGs can also take significant processing time in the browser.
* SVG can be harder to create than raster images, depending on what kind of image you are trying to create.

## Adding SVG to webpage

Pros

Quick, familiar image syntax with built-in text equivalent available in the alt attribute.

You can make the image into a hyperlink easily by nesting the <img> inside an [<a>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a) element.

The SVG file can be cached by the browser, resulting in faster loading times for any page that uses the image loaded in the future.

### <img> element

Text

Description automatically generated with low confidence

**Cons**

* You cannot manipulate the image with JavaScript.
* If you want to control the SVG content with CSS, you must include inline CSS styles in your SVG code. (External stylesheets invoked from the SVG file take no effect.)
* You cannot restyle the image with CSS pseudoclasses (like :focus)

### Troubleshooting

For browsers that don’t support SVG: you could reference a PNG or JPG from your src attribute and use a [srcset](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img#attr-srcset) attribute (which only recent browsers recognize) to reference the SVG

This being the case, only supporting browsers will load the SVG — older browsers will load the PNG instead:

Graphical user interface, text, application

Description automatically generated

### CSS Background Image

In the below code, older browsers will stick with the PNG that they understand, while newer browsers will load the SVG:

Text

Description automatically generated with low confidence

inserting SVGs using CSS background images means that the SVG can't be manipulated with JavaScript, and is also subject to the same CSS limitations.

### Inline SVG

You can also open up the SVG file in a text editor, copy the SVG code, and paste it into your HTML document

A picture containing text

Description automatically generated

Pros

Putting your SVG inline saves an HTTP request, and therefore can reduce your loading time a bit.

You can assign classes and ids to SVG elements and style them with CSS, either within the SVG or wherever you put the CSS style rules for your HTML document. In fact, you can use any SVG presentation attribute as a CSS property.

Inlining SVG is the only approach that lets you use CSS interactions (like :focus) and CSS animations on your SVG image (even in your regular stylesheet.)

You can make SVG markup into a hyperlink by wrapping it in an <a> element.

Cons

The browser cannot cache inline SVG as it would cache regular image assets, so pages that include the image will not load faster after the first page containing the image is loaded.

You may include fallback in a <foreignObject> element, but browsers that support SVG still download any fallback images. You need to weigh whether the extra overhead is really worthwhile, just to support obsolescent browsers.

### Iframe

You can open SVG images in your browser just like webpages

A picture containing text

Description automatically generated

Cons

* iframes do have a fallback mechanism, as you can see, but browsers only display the fallback if they lack support for iframes altogether.
* Moreover, unless the SVG and your current webpage have the same [origin](https://developer.mozilla.org/en-US/docs/Glossary/Origin), you cannot use JavaScript on your main webpage to manipulate the SVG.

# HTML Tables

## <table></table>

The point of a table is that it is rigid. Information is easily interpreted by making visual associations between row and column headers

When implemented correctly, HTML tables are handled well by accessibility tools such as screen readers, so a successful HTML table should enhance the experience of sighted and visually impaired users alike.

Be under no illusion; for tables to be effective on the web, you need to provide some styling information with [CSS](https://developer.mozilla.org/en-US/docs/Learn/CSS), as well as good solid structure with HTML

## <td></td> 🡪 a cell in the table

The smallest container inside a table is a table cell

A picture containing company name

Description automatically generated

If we want a row of four cells:

the cells are not placed underneath each other, rather they are automatically aligned with each other on the same row. Each <td> element creates a single cell and together they make up the first row. Every cell we add makes the row grow longer.

Text

Description automatically generated

## <tr></tr> 🡪 table row

Each row needs to be wrapped in an additional <tr> element, with each cell contained in a <td>

A screenshot of a computer

Description automatically generated with low confidence

## <th> </th> table header

table headers — special cells that go at the start of a row or column and define the type of data that row or column contains

Table headings come with some default styling — they are bold and centered even if you don't add your own styling to the table, to help them stand out.

Table header 🡪 they allow you to make tables more accessible by associating each header with all the data in the same row or column. Screen readers are then able to read out a whole row or column of data at once, which is pretty useful.

scope attribute

## [Allowing cells to span multiple rows and columns](https://developer.mozilla.org/en-US/docs/Learn/HTML/Tables/Basics#allowing_cells_to_span_multiple_rows_and_columns)

Table headers and cells have the colspan and rowspan attributes. Both accept a unitless number value, which equals the number of rows or columns you want spanned

## Styling entire Column

HTML has a method of defining styling information for an entire column of data all in one place — the [**<col>**](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/col) and [**<colgroup>**](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/colgroup) elements. These exist because it can be a bit annoying and inefficient having to specify styling on columns — you generally have to specify your styling information on every <td> or <th> in the column, or use a complex selector such as [:nth-child](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-child).

Styling columns like this is [limited to a few properties](https://www.w3.org/TR/CSS22/tables.html#columns): [border](https://developer.mozilla.org/en-US/docs/Web/CSS/border), [background](https://developer.mozilla.org/en-US/docs/Web/CSS/background), [width](https://developer.mozilla.org/en-US/docs/Web/CSS/width), and [visibility](https://developer.mozilla.org/en-US/docs/Web/CSS/visibility). To set other properties you'll have to either style every <td> or <th> in the column, or use a complex selector such as [:nth-child](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-child).

<col> elements are specified inside a <colgroup> container just below the opening <table> tag

Graphical user interface, text, application

Description automatically generated Table

Description automatically generated

Effectively we are defining two "style columns", one specifying styling information for each column. We are not styling the first column, but we still have to include a blank <col> element — if we didn't, the styling would just be applied to the first column.

If we wanted to apply the styling information to both columns, we could just include one <col> element with a span attribute on it, like this:

A picture containing logo

Description automatically generated

Just like colspan and rowspan, span takes a unitless number value that specifies the number of columns you want the styling to apply to

Could also use a css class

# Positioning

The **position** [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) property sets how an element is positioned in a document. The [top](https://developer.mozilla.org/en-US/docs/Web/CSS/top), [right](https://developer.mozilla.org/en-US/docs/Web/CSS/right), [bottom](https://developer.mozilla.org/en-US/docs/Web/CSS/bottom), and [left](https://developer.mozilla.org/en-US/docs/Web/CSS/left) properties determine the final location of positioned elements

## static

The element is positioned according to the normal flow of the document. The top, right, bottom, left, and z-index properties have no effect. This is the default value.

## relative

The element is positioned according to the normal flow of the document, and then offset relative to itself based on the values of top, right, bottom, and left. The offset does not affect the position of any other elements; thus, the space given for the element in the page layout is the same as if position were static.

This value creates a new stacking context when the value of z-index is not auto. Its effect on table-\*-group, table-row, table-column, table-cell, and table-caption elements is undefined.

absolute

The element is removed from the normal document flow, and no space is created for the element in the page layout. It is positioned relative to its closest positioned ancestor, if any; otherwise, it is placed relative to the initial containing block. Its final position is determined by the values of top, right, bottom, and left.

This value creates a new stacking context when the value of z-index is not auto. The margins of absolutely positioned boxes do not collapse with other margins.

fixed

The element is removed from the normal document flow, and no space is created for the element in the page layout. It is positioned relative to the initial containing block established by the viewport, except when one of its ancestors has a transform, perspective, or filter property set to something other than none (see the CSS Transforms Spec), or the will-change property is set to transform, in which case that ancestor behaves as the containing block. (Note that there are browser inconsistencies with perspective and filter contributing to containing block formation.) Its final position is determined by the values of top, right, bottom, and left.

This value always creates a new stacking context. In printed documents, the element is placed in the same position on every page.

sticky

The element is positioned according to the normal flow of the document, and then offset relative to its nearest scrolling ancestor and containing block (nearest block-level ancestor), including table-related elements, based on the values of top, right, bottom, and left. The offset does not affect the position of any other elements.

This value always creates a new stacking context. Note that a sticky element "sticks" to its nearest ancestor that has a "scrolling mechanism" (created when overflow is hidden, scroll, auto, or overlay), even if that ancestor isn't the nearest actually scrolling ancestor.

* A **stickily positioned element** is an element whose [computed](https://developer.mozilla.org/en-US/docs/Web/CSS/computed_value) position value is sticky. It's treated as relatively positioned until its [containing block](https://developer.mozilla.org/en-US/docs/Web/CSS/Containing_block) crosses a specified threshold (such as setting [top](https://developer.mozilla.org/en-US/docs/Web/CSS/top) to value other than auto) within its flow root (or the container it scrolls within), at which point it is treated as "stuck" until meeting the opposite edge of its [containing block](https://developer.mozilla.org/en-US/docs/Web/CSS/Containing_block).

Most of the time, absolutely positioned elements that have [height](https://developer.mozilla.org/en-US/docs/Web/CSS/height) and [width](https://developer.mozilla.org/en-US/docs/Web/CSS/width) set to auto are sized so as to fit their contents. However, non-[replaced](https://developer.mozilla.org/en-US/docs/Web/CSS/Replaced_element), absolutely positioned elements can be made to fill the available vertical space by specifying both [top](https://developer.mozilla.org/en-US/docs/Web/CSS/top) and [bottom](https://developer.mozilla.org/en-US/docs/Web/CSS/bottom) and leaving [height](https://developer.mozilla.org/en-US/docs/Web/CSS/height) unspecified (that is, auto). They can likewise be made to fill the available horizontal space by specifying both [left](https://developer.mozilla.org/en-US/docs/Web/CSS/left) and [right](https://developer.mozilla.org/en-US/docs/Web/CSS/right) and leaving [width](https://developer.mozilla.org/en-US/docs/Web/CSS/width) as auto.

Except for the case just described (of absolutely positioned elements filling the available space):

* If both top and bottom are specified (technically, not auto), top wins.
* If both left and right are specified, left wins when [direction](https://developer.mozilla.org/en-US/docs/Web/CSS/direction) is ltr (English, horizontal Japanese, etc.) and right wins when [direction](https://developer.mozilla.org/en-US/docs/Web/CSS/direction) is rtl (Persian, Arabic, Hebrew, etc.).

# CSS

Text

Description automatically generated

Example: If it is a paragraph, how should it look

Handles some interaction

# JavaScript

A picture containing text

Description automatically generated

Robust interaction like manipulating the data of a website