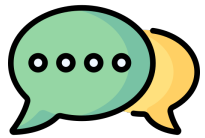
HYPERMEDIA APPLICATIONS (WEB AND MULTIMEDIA)

INTRODUCTION TO WEB DEVELOPMENT, HTML AND CSS

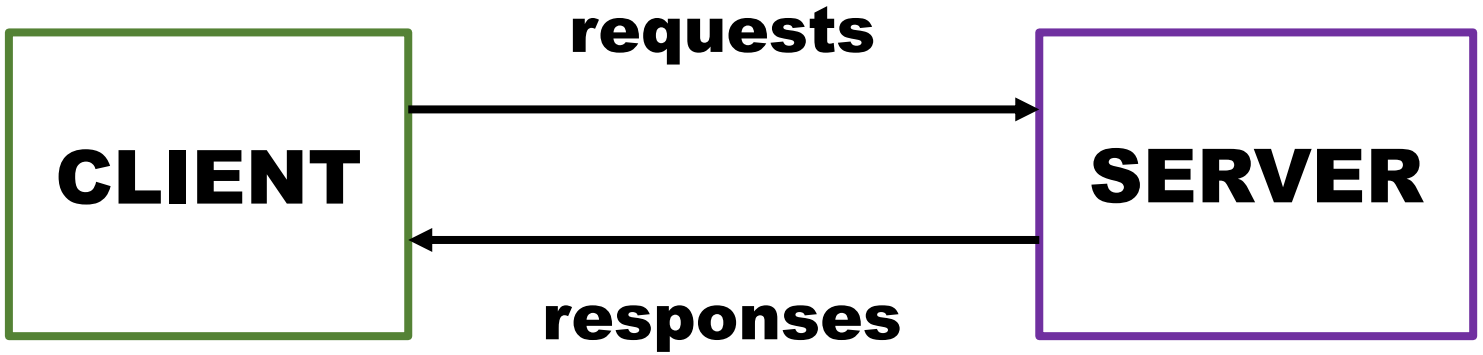
Francesco Vona

HOW MANY OF YOU KNOW WHAT HAPPEN WHEN YOU VISIT A WEBSITE?



Clients and Servers

Computers connected to the web are called *clients* and *servers*.
A **simplified** diagram of how they interact might look like this:



Clients

Clients are internet-connected devices that communicates with a server (e.g., your web-browser).

Whenever you access a website, your device is the client, and the web server hosting the website is the server.

Servers

Servers are computers that store webpages, sites, apps or just multimedia files.

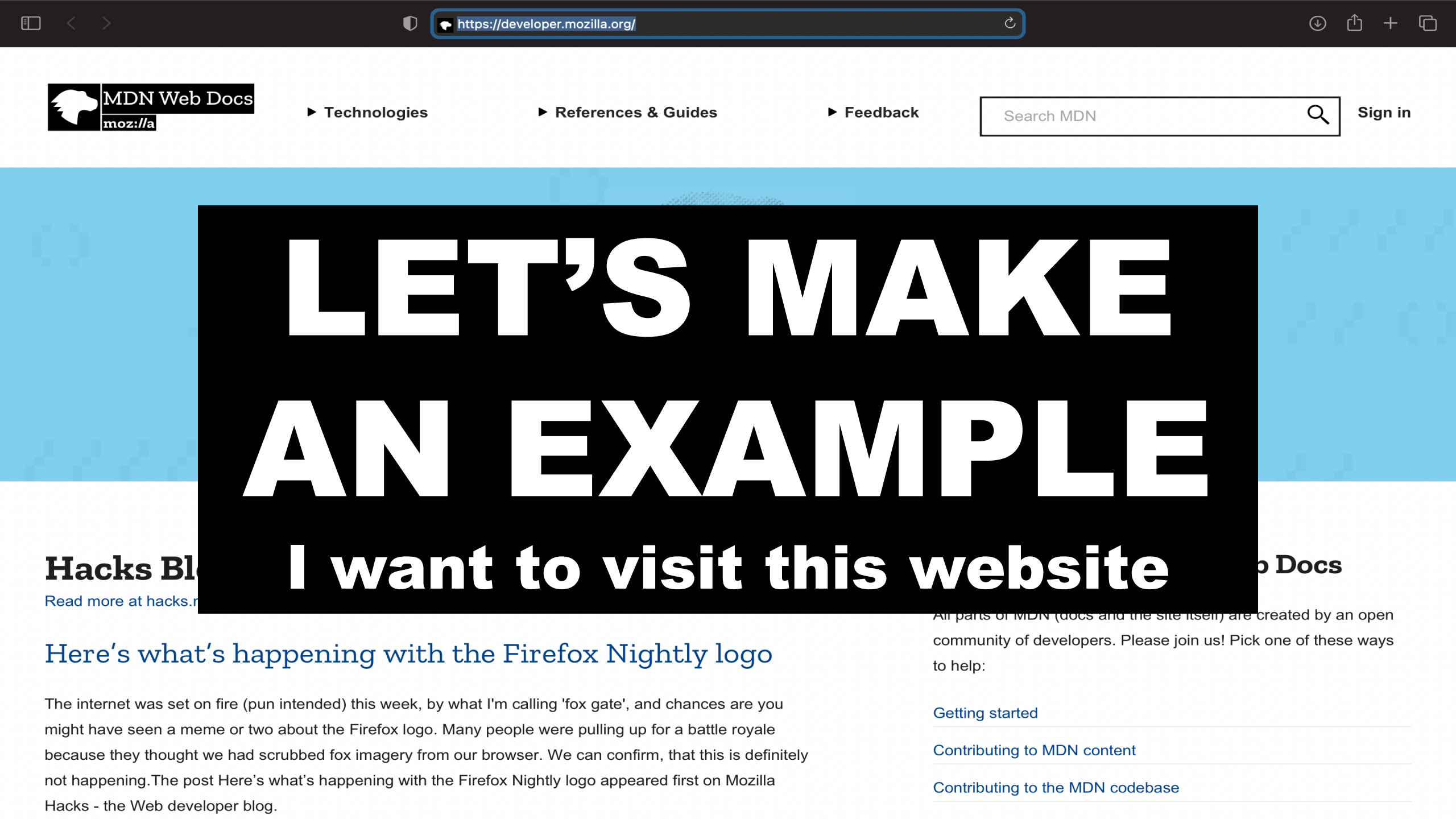
When a client device wants to access a webpage, a copy of the webpage is downloaded from the server onto the client machine to be displayed in the user's web browser.

HTTP – HyperText Transfer Protocol

HTTP is the protocol that rules how hypertext (data) gets transferred in a network.

A protocol is a set of rules that helps multiple parties understand how they have to implement with the process defined in the protocol.

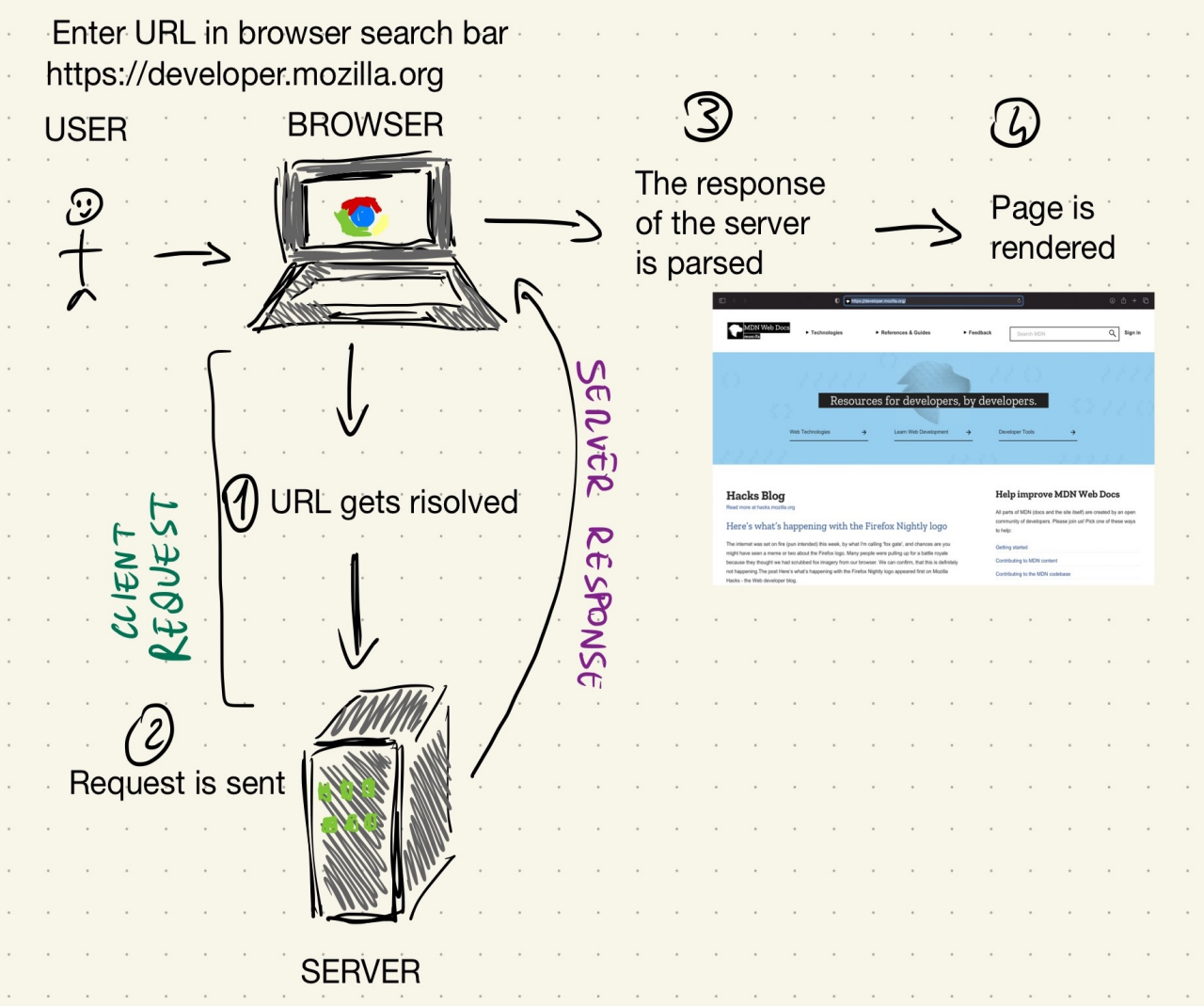
This means that HTTP already has the constructs needed in order to specify the action and the resource being acted upon (*client request*), as well as the outcome of those actions (*server response).*


https://developer.mozilla.org/
MDN Web Docs
moz://a
Technologies
References & Guides
Feedback
Search MDN
Sign in
LET'S MAKE AN EXAMPLE
I want to visit this website
Hacks Bl
Read more at hacks.
Here's what's happening with the Firefox Nightly logo
The internet was set on fire (pun intended) this week, by what I'm calling 'fox gate', and chances are you might have seen a meme or two about the Firefox logo. Many people were pulling up for a battle royale because they thought we had scrubbed fox imagery from our browser. We can confirm, that this is definitely not happening.The post Here's what's happening with the Firefox Nightly logo appeared first on Mozilla Hacks - the Web developer blog.
Docs
All parts of MDN (docs and the site itself) are created by an open community of developers. Please join us! Pick one of these ways to help:
Getting started
Contributing to MDN content
Contributing to the MDN codebase

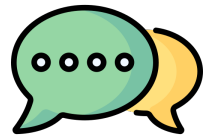
How the Web Works

In the moment you ENTER this address - the URL - in your browser many thing happen:

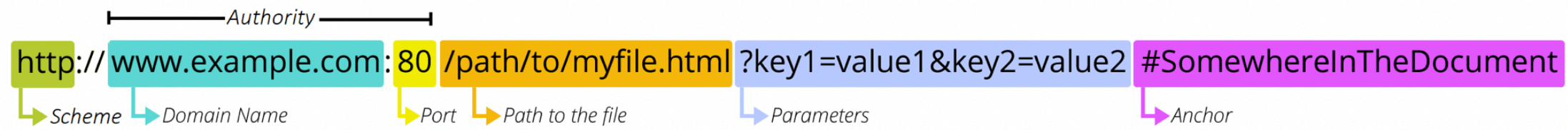
1. The URL gets resolved
2. A Request is sent to the server of the website
3. The response of the server is parsed by the browser
4. The page is rendered and displayed to the user



HOW MANY OF YOU KNOW WHAT URL STANDS FOR?



URL - Uniform Resource Locator

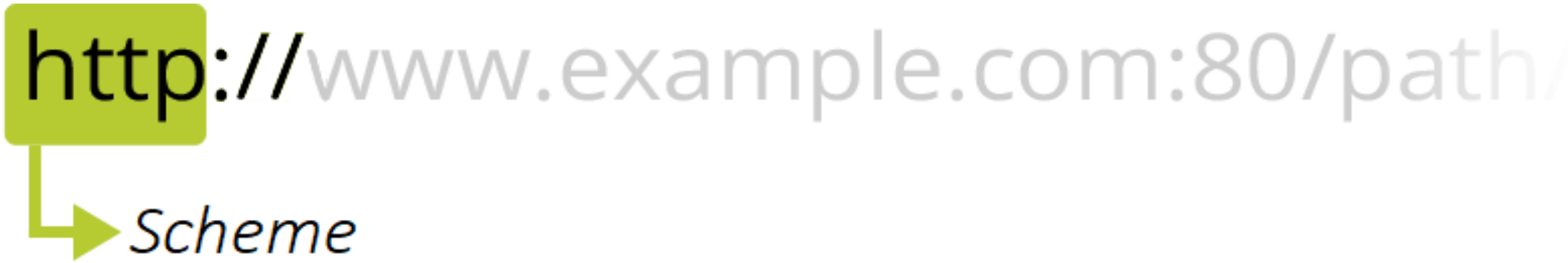


A URL is the address of a given unique resource on the Web.

It is used by browsers to retrieve resources.

Any URL can be typed right inside the browser's address bar to get to the resource behind it.

Scheme

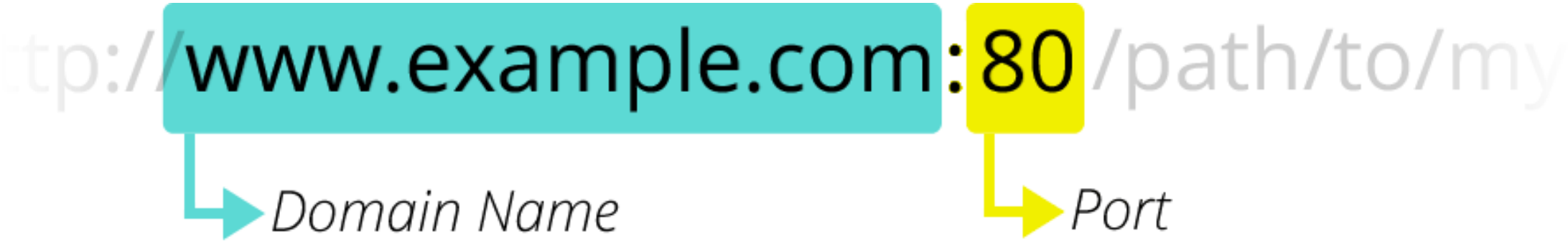


The scheme, indicates the protocol that the browser must use to request the resouce.

Usually for websites the protocol is HTTP or HTTPS (its secure version).

Browsers also know how to handle other schemes such as mailto: (to open a mail client) or ftp: to handle file transfer.

Authority



Next follows the authority which includes both the domain and the port, separated by a colon:

- The domain indicates which Web server is being requested.
- The port indicates the technical "gate" used to access the resources on the web server.
  It is usually omitted if the web server uses the standard ports of the HTTP or HTTPS protocol (80,443). Otherwise it is mandatory.

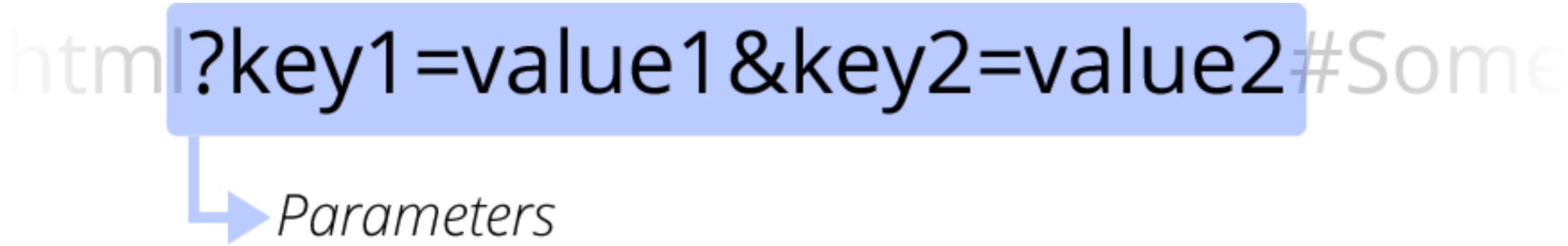
Path to resource

/path/to/myfile.html is the path to the resource on the Web server.

In the early days of the Web, a path like this represented a physical file location on the Web server.

Nowadays, it is mostly an abstraction handled by Web servers without any physical reality.

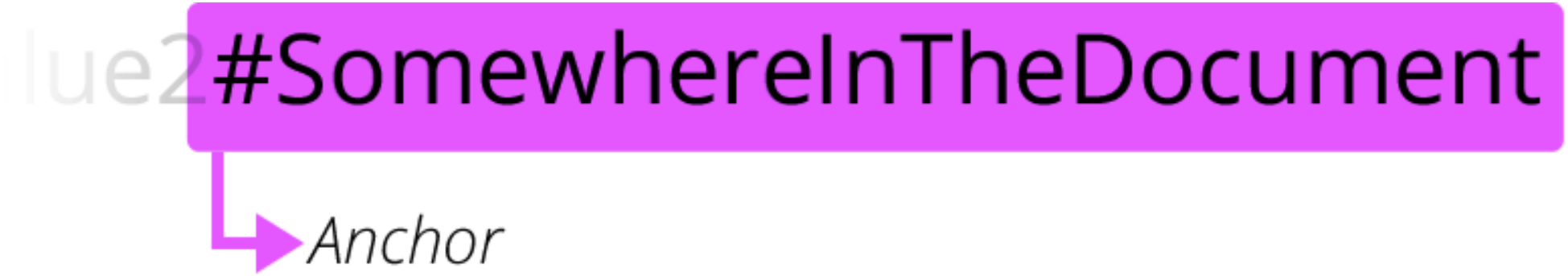
Parameters



Those parameters are a list of key/value pairs separated with the & symbol.

The Web server can use those parameters to do extra stuff before returning the resource. Each Web server has its own rules regarding parameters, and the only reliable way to know if a specific Web server is handling parameters is by asking the Web server owner.

Anchor



#SomewhereInTheDocument is an anchor to another part of the resource itself.

An anchor represents a sort of "bookmark" inside the resource.

It is worth noting that the part after the #, also known as the fragment identifier, is never sent to the server with the request.

URL Gets Resolved

You enter «developer.mozilla.org» but actually, real web addresses aren't like this.

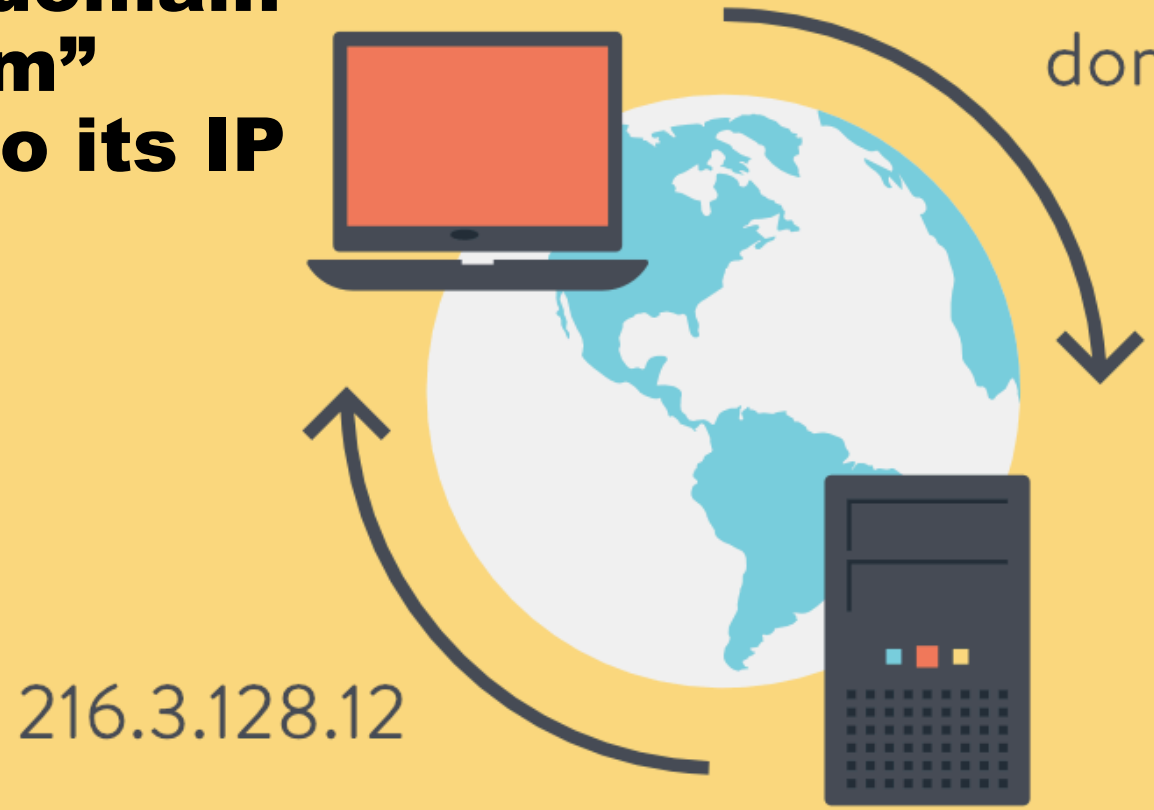
The server which hosts the source code of a website is identified via IP (= Internet Protocol) addresses indeed.

An IP address typically looks like this: 172.56.180.5 (IPv4)

Or like this: 2001:0db8:85a3:0000:1319:8a2e:0370:7344 (IPv6)

DNS – Domain Name System

How is the domain “domain.com” translated to its IP address?



There’s a special type of server called “name server” or “**DNS server**” (where DNS = “Domain Name System”) that translates domains to IP addresses.

Request is sent

With the IP address translated, the browser makes a request to the server with that IP address.

The data is sent via the “HyperText Transfer Protocol” (known as “HTTP”)

HTTP Request

Requests consists of the following elements:

- An HTTP method: GET, POST, PUT, DELETE
- The path of the resource to fetch
- The domain (here, developer.mozilla.org) and, optionally, the TCP port (for http the default is 80).
- The version of the HTTP protocol.
- Optional headers that convey additional information for the servers.
- Or a body, for some methods like POST, similar to those in responses, which contain the resource sent.

HTTP Response

That’s defined by web developers. In the end, a response has to be sent.

Responses consist of the following elements:

- The version of the HTTP protocol they follow.
- A status code, indicating if the request was successful, or not, and why.
- A status message, a non-authoritative short description of the status code.
- HTTP headers, like those for requests.
- Optionally, a body containing the fetched resource.

Response is Parsed

The browser receives the response sent by the server and parses the response. Just as the server did it with the request.

The browser checks the data and metadata that’s enclosed in the response (HTTP protocol). And based on that, it decides what to do.

In the case of our example, the response would contain a specific piece of metadata that tells the browser that the response data is of type text/html, i.e. a website.

Not only webpages

The core idea of requests and responses is always the same.

But not every response is necessarily a website and not every request wants a website.

The metadata which is attached to requests and responses controls which data is wanted and returned. Of course both parties that are involved (i.e. client and server) need to support the requests and sent data.

For example, you can’t request a PDF from google.com because Google doesn’t support this kind of requested data for this specific URL.

But there are many servers that specialize in providing URLs that return certain pieces of data. Such services are also referred to as “APIs” (“Application Programming Interface”).

Server-side vs Browser-side

From the four steps above, you learned that there are two core “sides” when talking about the web: Server-side and Client-side (or: Browser-side).

As a web developer, it’s important to know that you use different technologies and programming languages for these sides.

Server-side

Server-side programming languages don’t work in the browser but they can run on a normal computer.

Examples would be:

nodeJS

python

php

With the exception of PHP, you can also use these programming languages for other purposes than web development.

Client-side

In the browser, there are exactly three languages/ technologies you need to learn. But whilst the server-side languages were alternatives, these three browser-side languages are all mandatory to know and understand:

HTML
(for the structure)

CSS
(for the styling)

JavaScript
(for dynamic content)

HTML

HTML stands for “Hyper Text Markup Language” and it describes the structure of a webpage.

The code is composed by “HTML tags” such as:

<h1></h1> and

Every HTML tag has some semantic meaning which the browser understands, because HTML is also standardized.

CSS

CSS (“Cascading Style Sheets”) is all about adding styling to the website.

That is done via “CSS rules”.

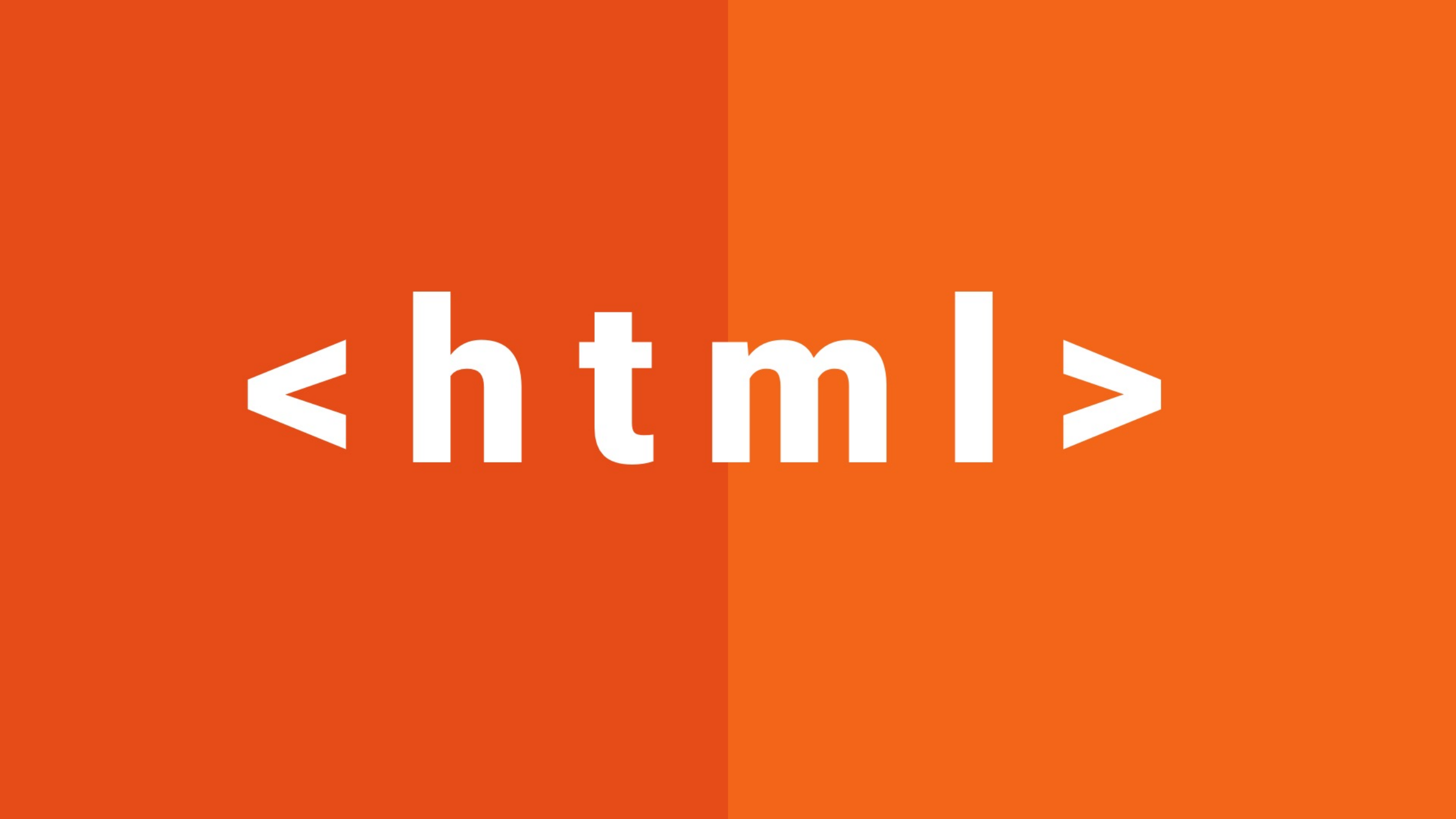
Rules are usually part of separate .css files which are requested separately.

JavaScript

JavaScript is the browser-side programming language!

It’s not always visible but all dynamic content you find on a website (e.g. tabs, overlays etc.) is actually only possible because of JavaScript.

It allows web developers to define code that runs in the browser.


<html>

HTML – HyperText Markup Language

HTML is a markup language (not a programming language).

HTML code tells the browser how to structure a webpage.

HTML uses "markup" to annotate text, images, and other content for display in a Web browser.

HTML markup includes special "elements" such as:

<head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, <ul>, <ol>, <li> and many others.

An example

```
My cat is very grumpy

<!--If we want that our browser interprests this text
as a paragraph we can enclose it between the <p> tag -->


<p>My cat is very grumpy</p>
```

HTML Comments

With HTML you can write comments in the code.

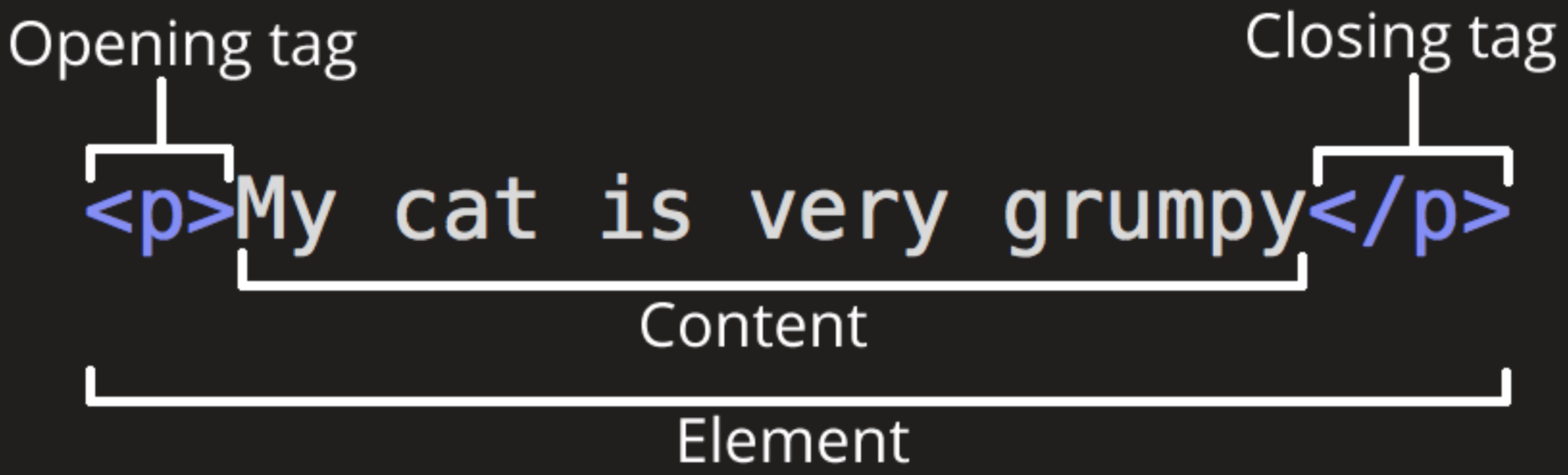
Browsers ignore comments, making them invisible to the user.

The purpose of comments is to allow you to include notes in the code to explain your logic or coding.

To write an HTML comment, wrap it in the special markers

<!-- and -->

Anatomy of an HTML element



Anatomy of an HTML element

The anatomy of our element is:

- The **opening tag**: This consists of the name of the element wrapped in opening and closing angle brackets.
- The **content**: This is the content of the element.
- The **closing tag**: This is the same as the opening tag, except that it includes a forward slash before the element name. This marks where the element ends.

The HTML element is: the opening tag, followed by content, followed by the closing tag.

Whitespace in HTML

```
<!-- No matter how much whitespace you use inside HTML element content
the HTML parser reduces each sequence of whitespace to a single space
when rendering the code. So why use so much whitespace?
To make your code much readable.-->

<p>My cat is very grunpy</p>
<p>     My      cat         is      very        grunpy      </p>
```

Nesting elements

```
<!-- Elements can be placed within other elements (nesting).
For example if in our paragraph <p>My cat is very grumpy</p>
we want to highlight a specific word we can use another element
nested inside the paragraph like this: -->

<p>My cat is <strong>very</strong> grumpy </p>

<!-- The tags have to open and close in a way that
they are inside or outside one another. -->
```

Block and Inline elements

There are two important categories of elements to know in HTML: block-level elements and inline elements.

HTML5 redefined the element categories. While these definitions are more accurate and less ambiguous than their predecessors, the new definitions are a lot more complicated to understand than block and inline.

Element content categories

Block elements

Block-level elements form a visible block on a page.

A block-level element appears on a new line.

Any content that follows a block-level element also appears on a new line.

Block-level elements are usually structural elements on the page (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.

Inline elements

Inline elements are contained within block-level elements, and surround only small parts of the document’s 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> element creates a hyperlink, and elements such as or create emphasis.

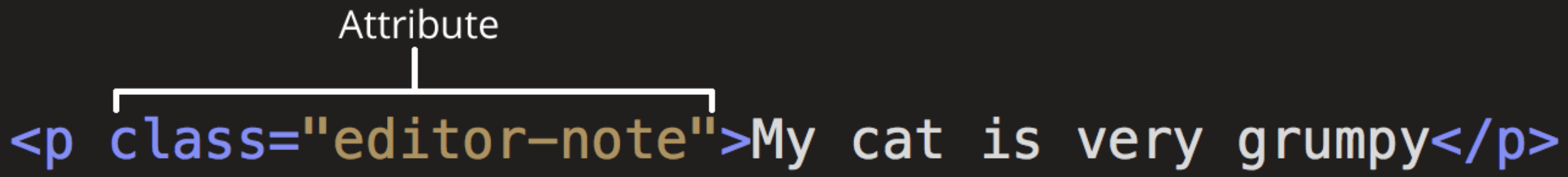
Empty elements

```
<!-- Not all elements follow the pattern of an opening tag,
content, and a closing tag. Some elements consist of a single tag,
which is typically used to insert/embed something in the document.
For example, the <img>element embeds an image file onto a page: -->

<img src="https://cdn.mos.cms.futurecdn.net/5FmFtc974AjN255w6iELLj.jpg">
```

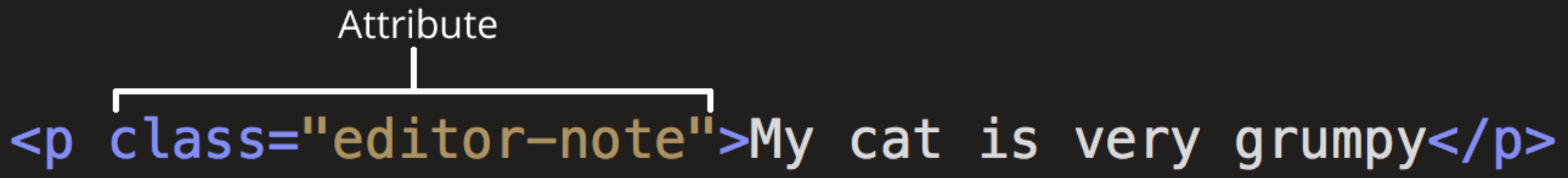
HTML elements

Attributes



Attributes contain extra information about the element that won't appear in the content.

Attributes



An attribute should have:

- **A space between it and the element name.**
- **The attribute name, followed by an equal sign.**
- **An attribute value, wrapped with opening and closing quote marks.**

Boolean Attributes

```
<!-- Sometimes you will see attributes written without values.
Boolean attributes can only have one value,
which is generally the same as the attribute name.
For example, consider the disabled attribute -->

<input type="text" disabled>
```

Single or double quotes?

```
<!-- You can feel free to choose which one you prefer.
Both of these lines are equivalent: -->

<a href="https://google.com"> Click here </a>
<a href='https://google.com'> Click here </a>

<!-- Make sure you don't mix single quotes and double quotes.
However, if you use one type of quote, you can include
the other type of quote inside your attribute values: -->

<a href="http://google.com" title="Isn't this fun?">Click here</a>

<!-- To use quote marks inside other quote marks of the same type
(single quote or double quote), use HTML entities. -->
```

Entity references

In HTML, the characters <, >,",' and & are special characters.

They are parts of the HTML syntax itself. So how do you include one of these special characters in your text?

You do this with character references. Each character reference starts with an ampersand (&), and ends with a semicolon (;).

Literal character	Character reference equivalent
< (less than)	<
> (greater than)	>
“ (quote)	"
‘ (apostrophe)	'
& (ampersand)	&

HTML attributes

Anatomy of an HTML document

An HTML page is the composition of multiple HTML elements

```
<!DOCTYPE html>
<html>

    <head>
        <title>My First Webpage</title>
    </head>

    <body>
        <h1>
            My First Webpage
        </h1>
        <p>This is a paragraph...</p>
    </body>

</html>
```

Anatomy of an HTML document

The doctype

It was meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML.

More recently, the doctype is a historical artifact that needs to be included for everything else to work right.

```
<!DOCTYPE html>
<html>

    <head>
        <title>My First Webpage</title>
    </head>

    <body>
        <h1>
            My First Webpage
        </h1>
        <p>This is a paragraph...</p>
    </body>

</html>
```

Anatomy of an HTML document

The <html> element.

It's the root element of the page.
All the content on the page is wrapped in this element.

```
<!DOCTYPE html>
<html>

    <head>
        <title>My First Webpage</title>
    </head>

    <body>
        <h1>
            My First Webpage
        </h1>
        <p>This is a paragraph...</p>
    </body>

</html>
```

Anatomy of an HTML document

The <head> element.

It 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.

```
<!DOCTYPE html>
<html>

    <head>
        <title>My First Webpage</title>
    </head>

    <body>
        <h1>
            My First Webpage
        </h1>
        <p>This is a paragraph...</p>
    </body>

</html>
```

Anatomy of an HTML document

The <body> element.

It contains all the content displayed on the webpage.

```
<!DOCTYPE html>
<html>

    <head>
        <title>My First Webpage</title>
    </head>

    <body>
        <h1>
            My First Webpage
        </h1>
        <p>This is a paragraph...</p>
    </body>

</html>
```

What’s in the head?

The head of an HTML document is the part that is not displayed in the web browser when the page is loaded.

It contains information such as the page <title>, links to CSS, links to custom favicons, and other metadata (data about the HTML, such as the author, and important keywords that describe the document).

Adding a title

```
<!-- The <title> element can be used to add a title to the document. -->

<!DOCTYPE html>
<html>
  <head>
    <title>My test page</title>
  </head>
  <body>
    <p>This is my page</p>
  </body>
</html>
```

Adding metadata

Metadata is data that describes data.

HTML has an "official" way of adding metadata to a document — the <meta> element.

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.

Specifying your document's character encoding

```
<!-- The charset meta specifies the document's character encoding.
utf-8 is a universal character set that includes pretty much any character
from any human language. -->

<!DOCTYPE html>
<html>
  <head>
    <title>My test page</title>
    <meta charset="utf-8">
  </head>
  <body>
    <p>This is my page</p>
  </body>
</html>
```

Adding an author and description

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

Specifying an author is beneficial in many ways:

- it is useful to be able to understand who wrote the page, if you have any questions about the content and you would like to contact them.
- Some content management systems have facilities to automatically extract page author information and make it available for such purposes.

Adding favicon

```
<!-- To further enrich your site design, you can add references to custom
icons
in your metadata, and these will be displayed in certain contexts
(bookmarks).-->

<!DOCTYPE html>
<html>
  <head>
    <title>My test page</title>
    <meta charset="utf-8">
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
  </head>
  <body>
    <p>This is my page</p>
  </body>
</html>
```

Applying CSS and JavaScript to HTML

```
<!-- CSS and JS can be applied to a web page using the <link> element
and the <script> element, respectively.-->

<!DOCTYPE html>
<html>
  <head>
    <title>My test page</title>
    <meta charset="utf-8">
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
    <link rel="stylesheet" href="my-css-file.css">
    <script src="my-js-file.js" defer></script>
  </head>
  <body>
    <p>This is my page</p>
  </body>
</html>
```

Setting the primary language of the document

```
<!-- The lang attribute, in the opening HTML tag, can be set to set the
language of yor page.
This is useful in many ways:
- SEO: Your HTML document will be indexed more effectively
by search engines if its language is set.
- Accessibility: it is useful to people with visual impairments
using screen readers (for example, the word "six" exists in both French and
English,
but is pronounced differently.) -->

<!DOCTYPE html>
<html>
  <head>
  </head>
  <body lang="en-US">
    <p>This is my page</p>
  </body>
</html>
```

Headings and paragraphs

Most structured text consists of headings and paragraphs.

In HTML, each paragraph has to be wrapped in a <p> element.

Each heading has to be wrapped in a heading element.

There are six heading elements: <h1>, <h2>, <h3>, <h4>, <h5>, and <h6>. Each element represents a different level of content in the document.

HTML structural hierarchy

Unordered lists

```
<!--Unordered lists are used to mark up lists of items
for which the order of the items doesn't matter.
Every unordered list starts off with a <ul> element;
this wraps around all the list items elements <li>.
 -->
<!DOCTYPE html>
<html>
  <head></head>
  <body lang="en-US">
    <ul>
        <li>milk</li>
        <li>eggs</li>
        <li>bread</li>
        <li>hummus</li>
    </ul>
  </body>
</html>
```

Ordered lists

```
<!--Ordered lists are lists in which the order of the items does matter.
The markup structure is the same as for unordered lists,
except that you have to wrap the list items in an <ol> element, rather
than <ul>.-->
<!DOCTYPE html>
<html>
  <head></head>
  <body lang="en-US">
    <ol>
        <li>Drive to the end of the road</li>
        <li>Turn right</li>
        <li>Go straight across the first two roundabouts</li>
        <li>Turn left at the third roundabout</li>
        <li>The school is on your right, 300 meters up the road</li>
    </ol>
  </body>
</html>
```

Nesting lists

```
<!--It is perfectly ok to nest one list inside another one.-->
<!DOCTYPE html>
<html>
  <head></head>
  <body lang="en-US">
    <ol>
        <li>Remove the skin from the garlic.</li>
        <li>Remove all the seeds and stalk from the pepper.
            <ul>
                <li>For a coarse "chunky" hummus, process it for a while.
</li>
                <li>For a smooth hummus, process it for a longer time.</li>
            </ul>
        </li>
    </ol>
  </body>
</html>
```

HTML lists

Emphasis and importance

In human language, we often emphasize certain words to alter the meaning of a sentence, and we often want to mark certain words as important.

HTML provides various semantic elements to allow us to mark up textual content.

Emphasis

```
<!--When we want to add emphasis in written language we tend to stress words
by putting them in italics.
In HTML we use the <em> (emphasis) element to mark up such instances.
These are recognized by screen readers and spoken out in a different tone of
voice.
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> element and
some CSS, or perhaps an <i> element.
-->

    <p>I am <em>glad</em> you weren't <em>late</em>.</p>
    <p>I am <i>glad</i> you weren't <i>late</i>.</p>
```

Importance

```
<!-- To emphasize important words, we tend to bold them in written language.
In HTML we use the <strong> (strong importance) element to mark up such
instances.
These are recognized by screen readers and spoken in a different tone of
voice.
Browsers style this as bold text by default, but you shouldn't use this tag
purely to get bold styling. To do that, you'd use a <span> element and some
CSS, or perhaps a <b> element.-->

<p>This liquid is <strong>highly toxic</strong></p>
<p>This liquid is <b>highly toxic</b></p>
```

Italic, bold, underline

HTML5 redefined <b>, <i>, and <u> with new semantic roles:

- <i> is used to convey a meaning traditionally conveyed by italic: foreign words, taxonomic designation, technical terms, a thought...
- <b> is used to convey a meaning traditionally conveyed by bold: key words, product names, lead sentence...
- <u> is used to convey a meaning traditionally conveyed by underline: proper name, misspelling...

What is a hyperlink?

Hyperlinks makes the Web *a web.*

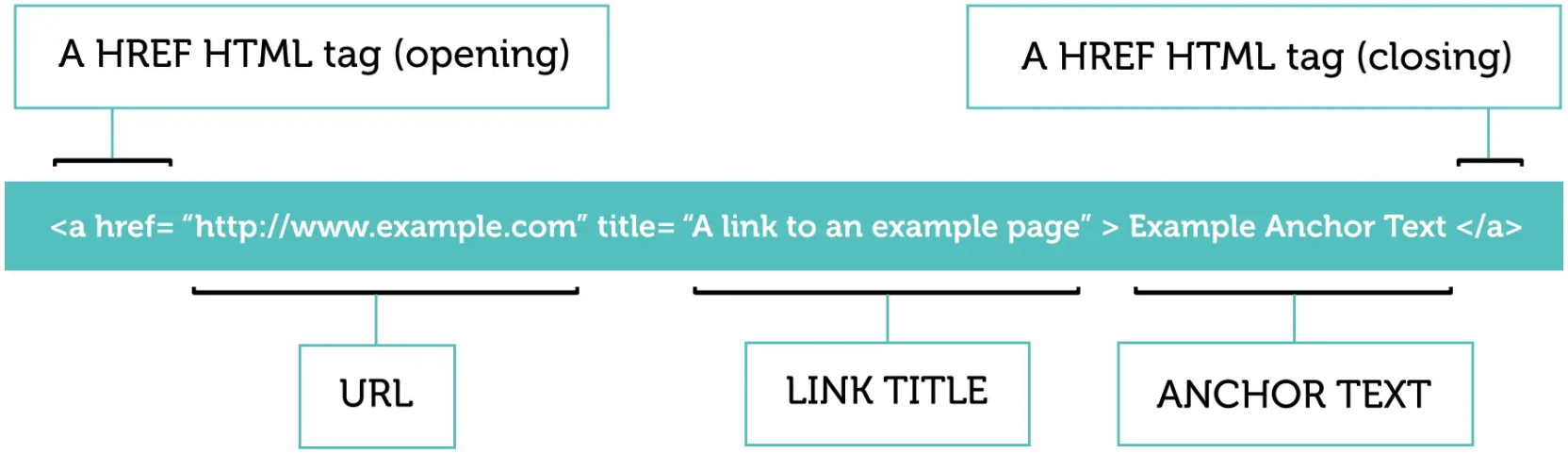
Hyperlinks allow us to link documents to other documents or resources, link to specific parts of documents, or make apps available at a web address.

Almost any web content can be converted to a link.

Anatomy of a link

A basic link is created by wrapping the text or other content inside an <a> element and using the href attribute that contains the web address.

The title contains additional information about the link.



Block level links

```
<!-- As mentioned before, almost any content can be made into a link,
even block-level elements.
If you have an image you want to make into a link, use the <a> element and
reference the image file with the <img>element.-->

<a href="https://www.mozilla.org/en-US/">
  <img src="mozilla-image.png" alt="mozilla logo that links to the mozilla
homepage">
</a>
```

Document and Website Strucutre

Webpages can and will look pretty different from one another, but they all tend to share similar standard components.

HTML layout elements

HTML provides dedicated tags that you can use to represent page sections, for example:

- **header:** <header>.
- **navigation bar:** <nav>.
- **main content:** <main>, with various content subsections represented by <article>, <section>, and elements.
- **sidebar:** <aside>; often placed inside <main>.
- **footer:** <footer>.

HTML layout elements in more detail

- <main> is for content *unique to this page.* Use <main> only *once* per page, and put it directly inside <body>. Ideally this shouldn't be nested within other elements.
- <article> encloses a block of related content that makes sense on its own without the rest of the page.
- <section> is similar to <article>, but it is more for grouping together a single part of the page that constitutes one single piece of functionality. It's considered best practice to begin each section with a heading.
- <aside> contains content that is not directly related to the main content but can provide additional information indirectly related to it.

HTML layout elements in more detail

- <header> represents a group of introductory content. If it is a child of <body> it defines the global header of a webpage, but if it's a child of an <article> or <section> it defines a specific header for that section.
- <nav> contains the main navigation functionality for the page. Secondary links, etc., would not go in the navigation.
- <footer> represents a group of end content for a page.

HTML layout elements

Non-semantic wrappers

Sometimes you'll come across a situation where you can't find an ideal semantic element to group some items together or wrap some content.

Sometimes you might want to just group a set of elements together to affect them all as a single entity with some CSS or JavaScript.

For cases like these, HTML provides the and elements. You should use these preferably with a suitable class attribute, to provide some kind of label for them so they can be easily targeted.


```
<!--<span> is an inline non-semantic element, which you should only use if you can't think of a better semantic text element to wrap your content, or don't want to add any specific meaning. For example:-->


<p>The King walked drunkenly back to his room at 01:00, the beer doing nothing to aid him as he staggered through the door <span class="editor-note">[Editor's note: At this point in the play, the lights should be down low]</span>.</p>
```



```
<!-- <div> is a block level non-semantic element, which you should only use
if you can't think of a better semantic block element to use, or don't want
to add any specific meaning. For example, -->

<div class="shopping-cart">
  <h2>Shopping cart</h2>
  <ul>
    <li>
      <p>
        <a href=""><strong>Silver earrings</strong></a>
        : $99.95.
      </p>
      <img src="../products/3333-0985/thumb.png" alt="Silver earrings">
    </li> <li> ... </li>
  </ul>
  <p>Total cost: $99.95</p>
</div>
```

Line breaks and horizontal rules

 creates a line break in a paragraph; it is the only way to force a rigid structure in a situation where you want a series of fixed short lines.

<hr> elements create a horizontal rule in the document that denotes a thematic change in the text (such as a change in topic or scene). Visually it just looks like a horizontal line.

HTML and debugging

HTML is not as complicated to understand as a real programming language like Javascript. HTML is not compiled into a different form before the browser parses it but it is *interpreted.*

Thus the way that browsers parse HTML is a lot more **permissive** than how programming languages are run, which is both a good and a bad thing.

So what do we mean by permissive? Well, there are two main types of error that you'll come across:

- **Syntax errors**: These are spelling errors in your code that actually cause the program not to run.
- **Logic errors**: These are errors where the syntax is actually correct, but the code is not what you intended it to be, meaning that the program runs incorrectly.

HTML itself doesn't suffer from syntax errors because browsers parse it permissively, meaning that the page still displays even if there are syntax errors.

Interpreting the error messages

Images in HTML

In order to put a simple image on a webpage, we use the <img> element.

This is an empty element(meaning that it has no text content or closing tag) that requires a minimum of one attribute to be useful — src (sometimes spoken as its full title, *source*).

The src attribute contains a path pointing to the image you want to embed in the page, which can be a relative or absolute URL, in the same way as href attribute values in <a> elements.

alt, width and height

```
<!-- You can use the width and height attributes to specify the width and
height of your image.
 You can use the alt attribute to display a textual description of the
image, in case the image cannot be seen/displayed or takes a long time to
render because of a slow internet connection-->

<img src="images/dinosaur.jpg" alt="The head and torso of a dinosaur
skeleton; it has a large head with long sharp teeth" width="400"
height="341">
```

Image titles

```
<!--As with links, you can also add title attributes to images, to provide
further supporting  information if needed. In our example, we could do this:
-->

<img src="images/dinosaur.jpg" alt="The head and torso of a dinosaur
skeleton; it has a large head with long sharp teeth" width="400"
height="341" title="A T-Rex on display in the Manchester University Museum">
```

Annotating images with figures and figure captions

```
<!-- Speaking of captions, there are a number of ways that you could add a caption to go with your image. For example, there would be nothing to stop you from doing this:-->

<div class="figure"> <img src="images/dinosaur.jpg" alt="The head and torso of a dinosaur skeleton; it has a large head with long sharp teeth" width="400" height="341"> <p>A T-Rex on display in the Manchester University Museum.</p> </div>

<!-- A better solution, is to use the
HTML5 <figure> and <figcaption> elements. These are created for exactly this purpose. Our above example could be rewritten like this: -->

<figure> <img src="images/dinosaur.jpg" alt="The head and torso of a dinosaur skeleton; it has a large head with long sharp teeth" width="400" height="341"> <figcaption>A T-Rex on display in the Manchester University Museum.</figcaption> </figure>
```

Video and Audio in HTML

```
<!-- The <video> element allows you to embed a video very easily. A really simple example looks like this: -->

<video src="rabbit320.webm" controls> <p>Your browser doesn't support HTML5 video. Here is a <a href="rabbit320.webm">link to the video</a> instead.</p> </video>

<!-- The <audio> element works just like the <video> element, with a few small differences as outlined below. A typical example might look like so: -->

<audio controls> <source src="viper.mp3" type="audio/mp3"> <source src="viper.ogg" type="audio/ogg"> <p>Your browser doesn't support HTML5 audio. Here is a <a href="viper.mp3">link to the audio</a> instead.</p> </audio>
```

HTML multimedia and embedding


CSS

CSS

CSS (Cascading Style Sheets) is used to style and lay out web pages.

CSS can be used for very basic document text styling — for example changing the color and size of headings and links.

It can be used to create layout — for example turning a single column of text into a layout with a main content area and a sidebar for related information.

It can even be used for effects such as animation.

CSS is a rule-based language — you define rules specifying groups of styles that should be applied to particular elements or groups of elements on your web page.

Applying CSS to HTML – External stylesheet

You reference an external CSS stylesheet from an HTML <link> element.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>
```

Applying CSS to HTML – Internal stylesheet

To create an internal stylesheet, you place CSS inside a <style> element contained inside the HTML <head>.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
    <style>
      p {
        color: red;
      }
    </style>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>
```

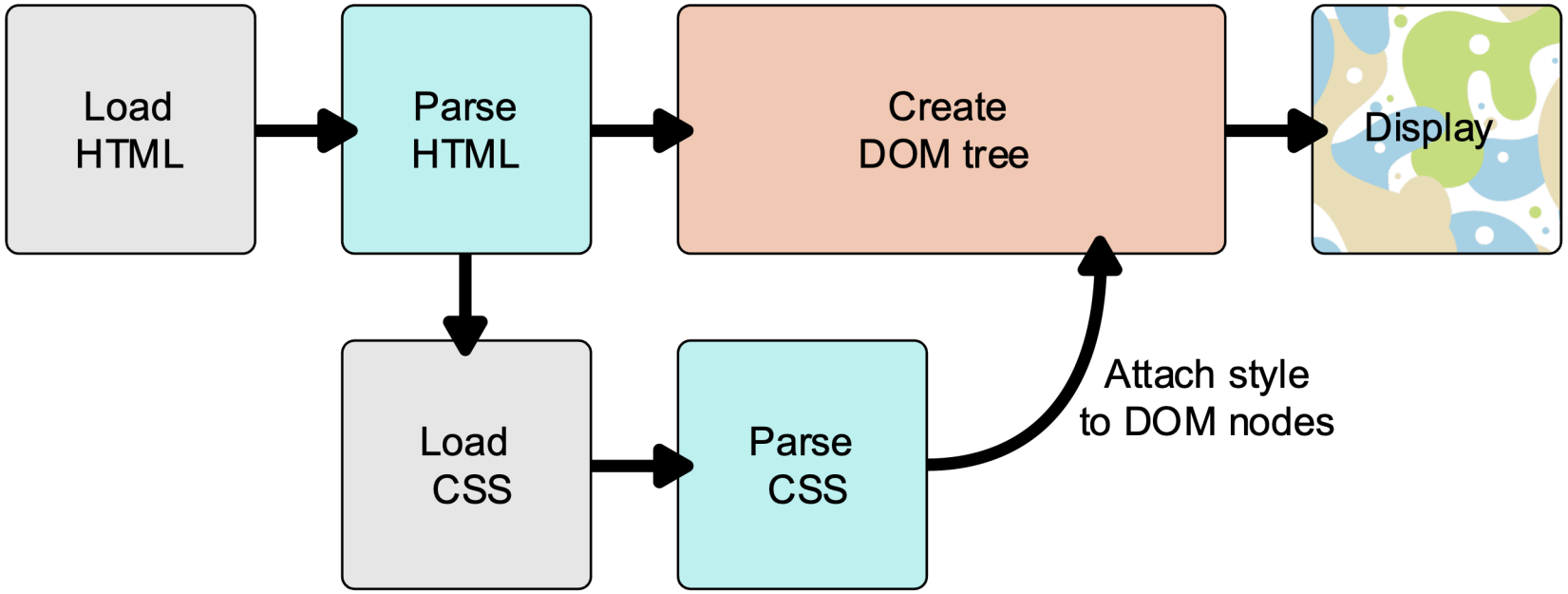
Applying CSS to HTML – Inline styles

Avoid using CSS in this way, when possible.
It is the opposite of a best practice.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
  </head>
  <body>
    <h1 style="color: blue;background-color: yellow;border: 1px solid
black;">Hello World!</h1>
    <p style="color:red;">This is my first CSS example</p>
  </body>
</html>
```

How does CSS actually work?

When a browser displays a document, it must combine the document's content with its style information.



The DOM

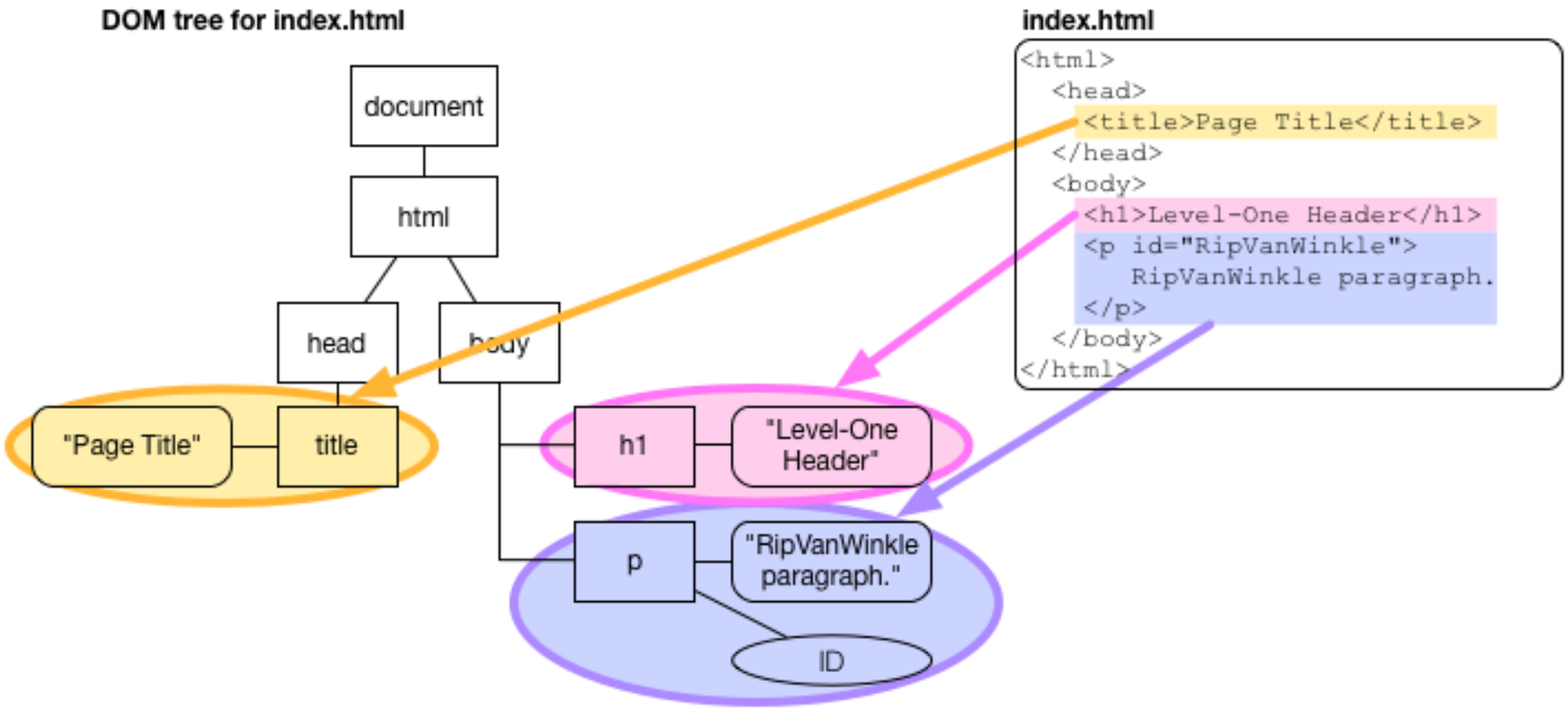
The DOM (Document Object Model) represents the document in the computer's memory.

A DOM has a tree-like structure. Each element, attribute, and piece of text in the markup language becomes a DOM node in the tree structure.

The nodes are defined by their relationship to other DOM nodes. Some elements are parents of child nodes, and child nodes have siblings.

The DOM is where your CSS and the document's content meet up.

DOM Example


DOM tree for index.html
document
html
head
body
"Page Title"
title
h1
"Level-One Header"
p
"RipVanWinkle paragraph."
ID
index.html
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>Level-One Header</h1>

RipVanWinkle paragraph.

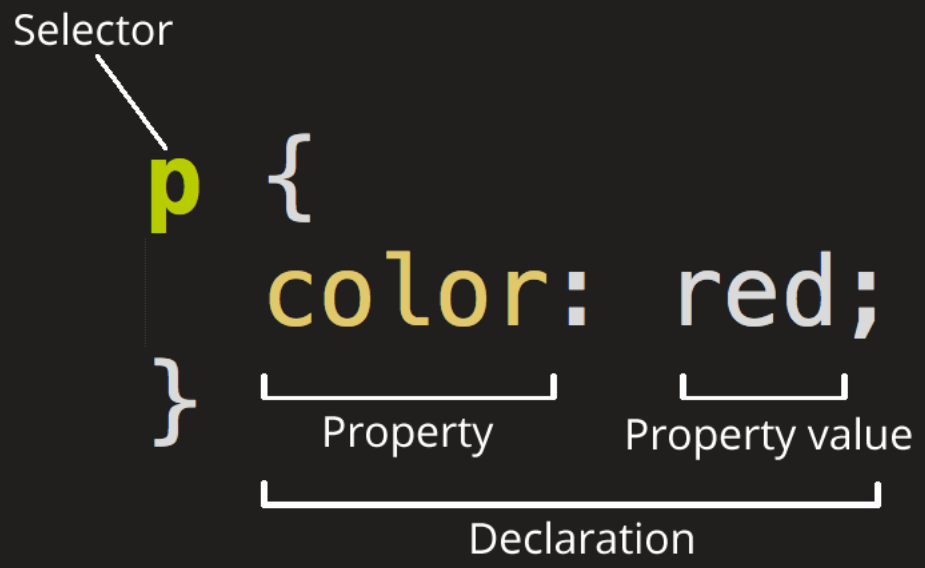
</body>
</html>

Anatomy of a CSS rule

The selector determines which HTML element we are going to style.

We then have a set of curly braces { }.

Inside those there will be one or more declarations, which take the form of property and value pairs.



CSS selectors

CSS selectors define the elements to which a set of CSS rules apply.

- Basic Selectors
- Grouping Selectors
- Combinators
- Pseudo (classes, elements)

Basic selectors

- **Universal** selector: Selects all elements. Optionally, it may be restricted to a specific namespace or to all namespaces.
  Syntax: * *ns*|* *|*

- **Type** selector: Selects all elements that have the given node name.
  Syntax: *elementname*

- **Class** selector: Selects all elements that have the given class attribute.
  Syntax: .*classname*

- **ID** selector: Selects an element based on the value of its id attribute. There should be only one element with a given ID in a document.
  Syntax: *#idname*

- **Attribute** selector: Selects all elements that have the given attribute.
  Syntax: [*attr*] [*attr=value*] [*attr~=value*] [*attr|=value*] [*attr^=value*][*attr$=value*] [*attr*=value*]

Grouping selectors

Selector list

The , is a grouping method that selects all the matching nodes.

Syntax: *A*, *B*
Example: div, span will match both and elements.

Combinators

- Descendant combinator
  The ' ' (space) combinator selects nodes that are descendants of the first element.
  Syntax: *A B*

- Child combinator
  The > combinator selects nodes that are direct children of the first element.
  Syntax: *A* > *B*

- General sibling combinator
  The ~ combinator selects siblings. This means that the second element follows the first (though not necessarily immediately), and both share the same parent.
  Syntax: *A* ~ *B*

- Adjacent sibling combinator
  The + combinator selects adjacent siblings. This means that the second element directly follows the first, and both share the same parent.
  Syntax: *A* + *B*

- Column combinator
  The || combinator selects nodes which belong to a column.
  Syntax: *A* || *B*

Pseudo selectors

- Pseudo classes
  The *:pseudo* allow the selection of elements based on state information that is not contained in the document tree.
  Example: a:visited will match all <a> elements that have been visited by the user.
- Pseudo elements
  The *::pseudo* represent entities that are not included in HTML.
  Example: p::first-line will match the first line of all <p> elements.

Link to resource

CSS Selectors

CSS conflicts

You may encounter scenarios where two selectors select the same HTML element.

The CSS language has rules to control which selector is stronger in the event of a conflict.

These rules are called **cascade** and **specificity.**

Cascade

The paragraph text will be blue.
This is because later styles replace conflicting styles that appear earlier in the stylesheet.
This is the cascade rule.

```
p {
  color: red;
}

p {
  color: blue;
}
```

Specificity

The paragraph text will be red.
A class is rated as being more specific, as in having more specificity than the element selector.

```
.special {
  color: red;
}

p {
  color: blue;
}
```

!important

!important is a special piece of CSS that you can use to overrule all of the above calculations.
This is used to make a particular property and value the most specific thing, thus overriding the normal rules of the cascade.

```css
#winning {
    background-color: red;
    border: 1px solid black;
}

.better {
    background-color: gray;
    border: none !important;
}

p {
    background-color: blue;
    color: white;
    padding: 5px;
}
```

The box model

Everything in CSS has a box around it, and understanding these boxes is key to being able to create layouts with CSS, or to align items with other items.

In CSS we broadly have two types of boxes — **block boxes** and **inline boxes**. These characteristics refer to how the box behaves in terms of page flow, and in relation to other boxes on the page.

The type of box applied to an element is defined by display property values such as block and inline, and relates to the **outer** value of display.

Boxes also have an *inner* display type, however, which dictates how elements inside that box are laid out. By default, the elements inside a box are laid out in **normal flow**

Parts of a box

Content box: The area where your content is displayed, which can be sized using properties like width and height.

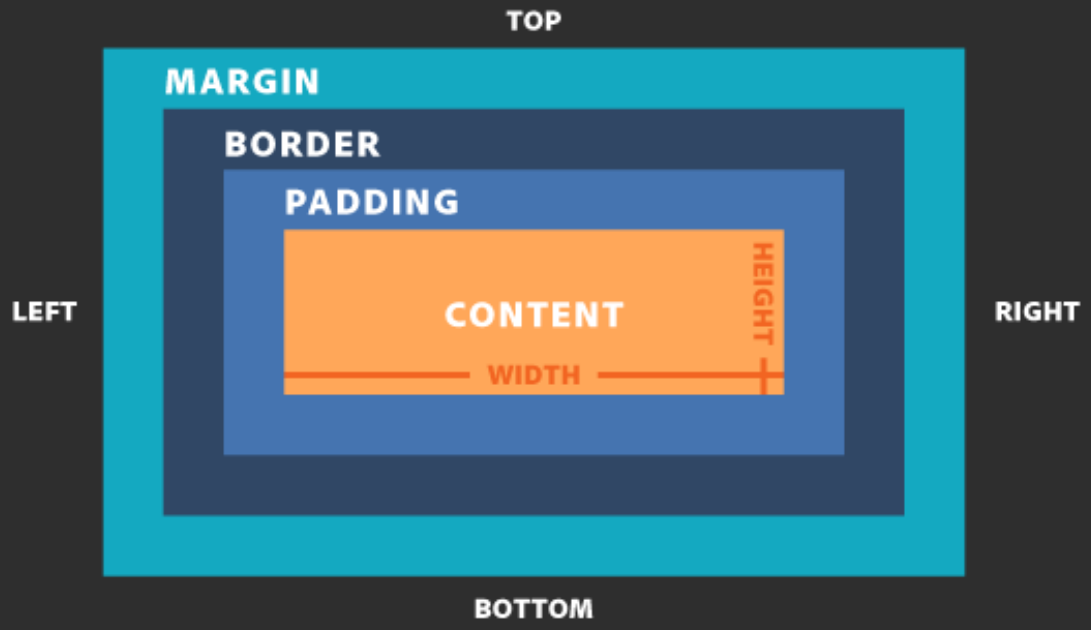
Padding box: The padding sits around the content as white space; its size can be controlled using padding and related properties.

Border box: The border box wraps the content and any padding. Its size and style can be controlled using border and related properties.

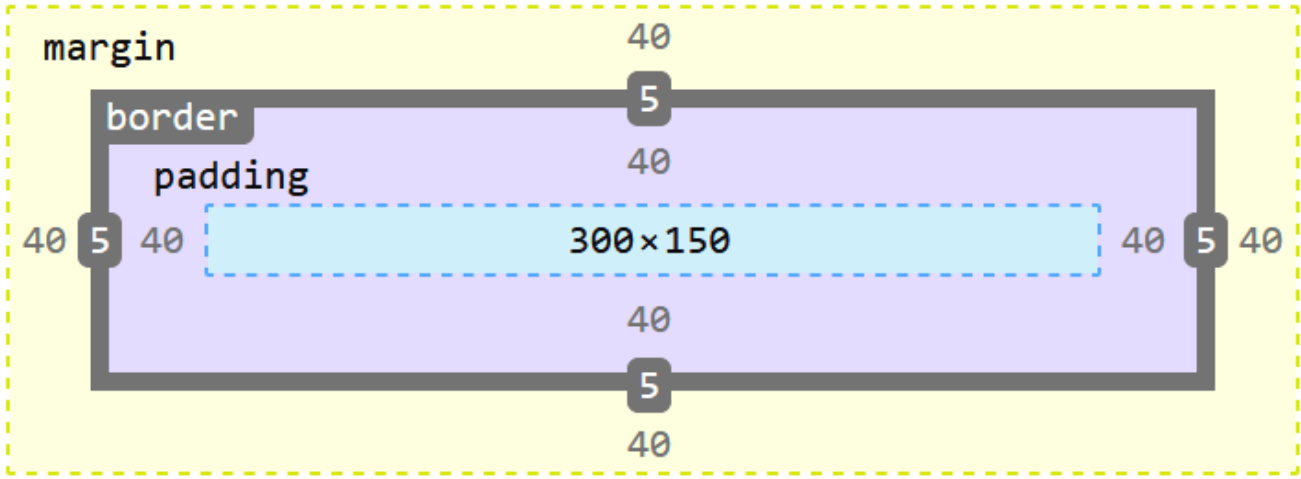
Margin box: The margin is the outermost layer, wrapping the content, padding and border as whitespace between this box and other elements. Its size can be controlled using margin and related properties.

CSS box model

The margin is not counted towards the actual size of the box.
Sure, it affects the total space that the box will take up on the page, but only the space outside the box.
The box's area stops at the border — it does not extend into the margin.



Use browser DevTools to view the box model



390×240 static

▼ Box Model Properties

box-sizing	content-box	line-height	28.8px
display	block	position	static
float	none	z-index	auto

Display: inline-block

An element with display: inline-block does a subset of the block things:

- The width and height properties are respected.
- padding, margin, and border will cause other elements to be pushed away from the box.

It does not, however, break onto a new line, and will only become larger than its content if you explicitly add width and height properties.

CSS box model

Overflowing content

Everything in CSS is a box. **Overflow happens when there is too much content to fit in a box.**

The overflow property is how you take control of an element's overflow.

The default value of overflow is *visible*.

With this default, we can see content when it overflows.

To crop content when it overflows, you can set overflow: *hidden*.

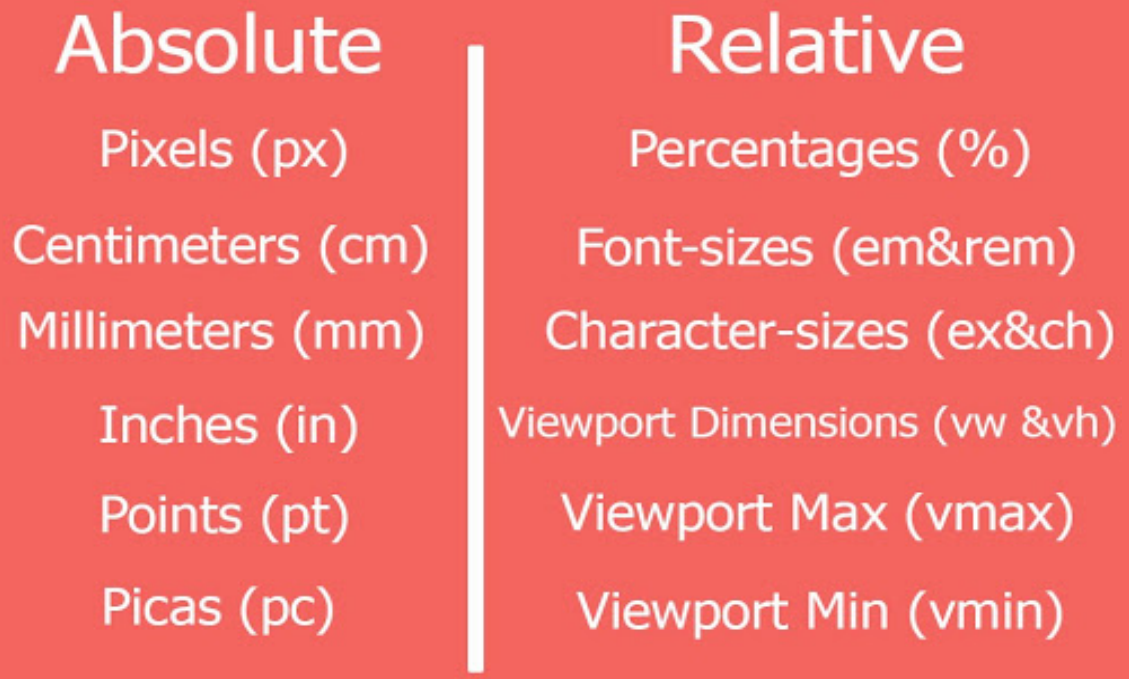
To add scrollbars when content overflows, use overflow: *scroll*

CSS overflow

CSS units

Absolute length units have a fixed size.

Relative length units are relative to a parent or the viewport, and style sheets written with relative units can more easily scale from one environment to another.



CSS Colors

There are many ways to specify color in CSS, some of which are more recently implemented than others. The same color values can be used everywhere in CSS.

Hexadecimal

RGB / RGBA

HSL / HSLA

String

```
.one {
  background-color: #02798b;
}

.two {
  background-color: rgb(2, 121, 139);
}

.three {
  background-color: rgba(2, 121, 139, .3);
}

.four {
  background-color: hsl(188, 97%, 28%);
}

.five {
  background-color: hsla(174, 77%, 31%, .9);
}

.six {
  background-color: red;
}
```

Sizing items in CSS

HTML Elements have a natural size, set before they are affected by any CSS.

An image has a width and a height defined in the image file (intrinsic size).

An empty , on the other hand, has no size of its own.

We can, of course, give elements in our design a specific size. When a size is given to an element (the content of which then needs to fit into that size) we refer to it as an **extrinsic size**.

Sizing items in CSS

- **Percentages**. When using a percentage you need to be aware what it is a percentage *of*. In the case of a box inside another container, if you give the child box a percentage width it will be a percentage of the width of the parent container.
- **min- and max- sizes.** In addition to giving things a fixed size, we can ask CSS to give an element a minimum or a maximum size.
- **Viewport** — which is the visible area of your page in the browser you are using to view a site — also has a size. In CSS we have the vw unit for viewport width, and vh for viewport height.

CSS sizing items

Styling text in CSS

The CSS properties used to style text generally fall into two categories:

- **Font styles**: Properties that affect the font that is applied to the text (color, font-family, font-style, font-weight, text-transform, text-decoration).
- **Text layout styles**: Properties that affect the spacing and other layout features of the text (text-align, line-height, letter-spacing, word-spacing).

CSS Layout

CSS page layout techniques allow us to take elements contained in a web page and control where they are positioned relative to their default position in normal layout flow.

Normal flow is how the browser lays out HTML pages by default when you do nothing to control page layout.

The methods that can change how elements are laid out in CSS are as follows: the display property, floats, the position property

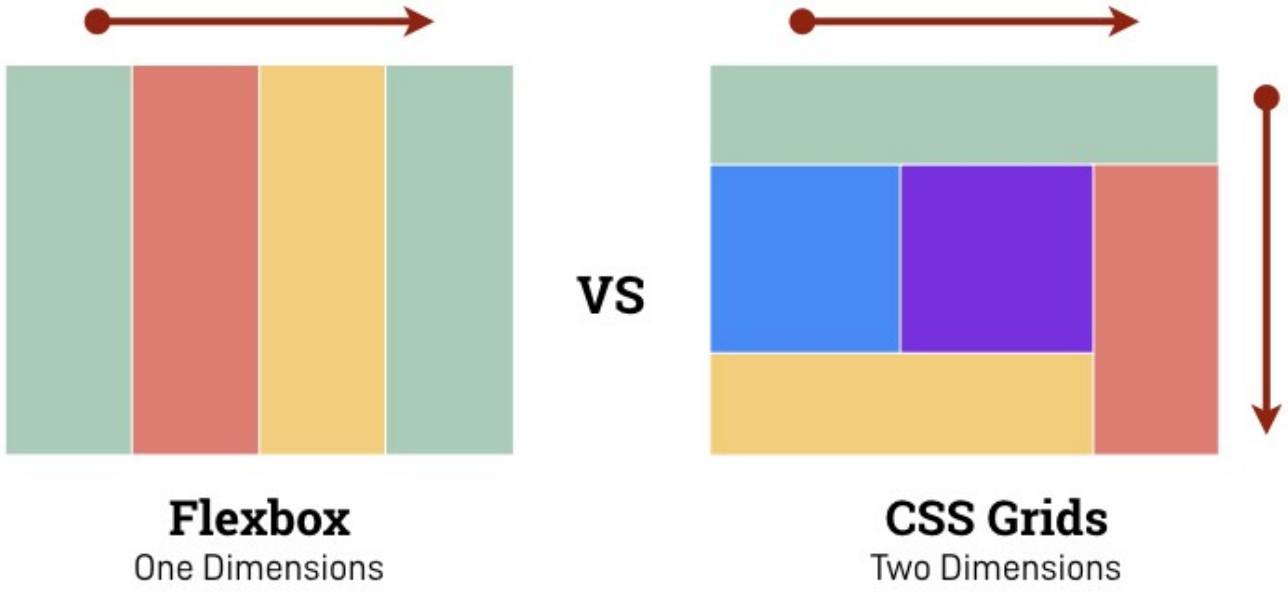
Display property

```
div {
  display: block; /* elements display on a new line */
  display: inline; /* elements display on the same line, no matter of height
and width*/
  display: inline-block; /* elements display on the same line, but sizes
matter*/
  display: none; /* hide */
  display: flex; /* display flexbox */
  display: grid; /* display gridbox */
}
```

display: flex; display: grid;

Flexbox is designed to make it easy for us to lay things out in one dimension — either as a row or as a column.

While flexbox is designed for one-dimensional layout, Grid Layout is designed for two dimensions — lining things up in rows and columns.



CSS display:flex and display:grid

Floats

Floating an element changes the behavior of that element and the block level elements that follow it in normal flow.

The element is moved to the left or right and removed from normal flow, and the surrounding content floats around the floated item.

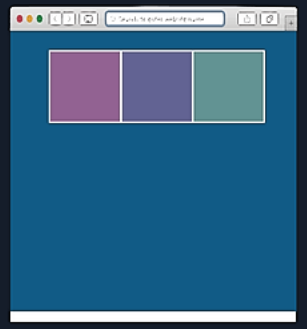
- left — Floats the element to the left.
- right — Floats the element to the right.
- none — Specifies no floating at all. This is the default value.
- inherit — Specifies that the value of the float property should be inherited from the element's parent element.

Positioning

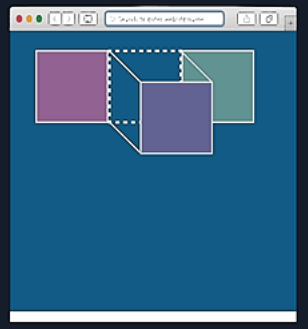
There are five types of positioning you should know about:

- **Static positioning** is the default that every element gets.
- **Relative positioning** allows you to modify an element's position on the page, moving it relative to its position in normal flow — including making it overlap other elements on the page.
- **Absolute positioning** moves an element completely out of the page's normal layout flow, like it is sitting on its own separate layer. From there, you can fix it in a position relative to the edges of the page's <html> element.
- **Fixed positioning** is very similar to absolute positioning, except that it fixes an element relative to the browser viewport, not another element.
- **Sticky positioning** is a newer positioning method which makes an element act like position: static until it hits a defined offset from the viewport, at which point it acts like position: fixed.

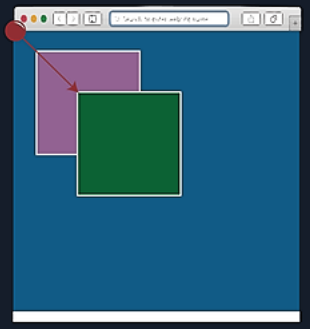
Positioning



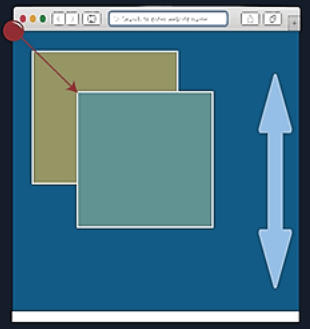
Static



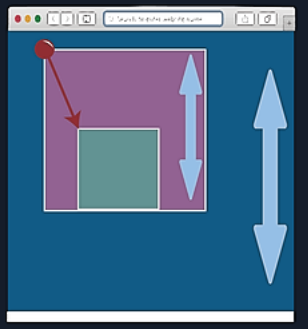
Relative



Absolute



Fixed



Sticky

CSS floats and positioning

Responsive Design

It is important to understand that **responsive web design isn't a separate technology** — it is a term used to describe an approach to web design or a set of best practices, used to create a layout that can *respond* to the device being used to view the content.

We already seen techniques to achieve a responsive design (viewport sizing, grid layout, flex layout, percentages) but responsive design was only able to emerge due to the media query.

Media queries let you adapt your site or app depending on the presence or value of various device characteristics and parameters.

CSS media-queries