

# Web Dev



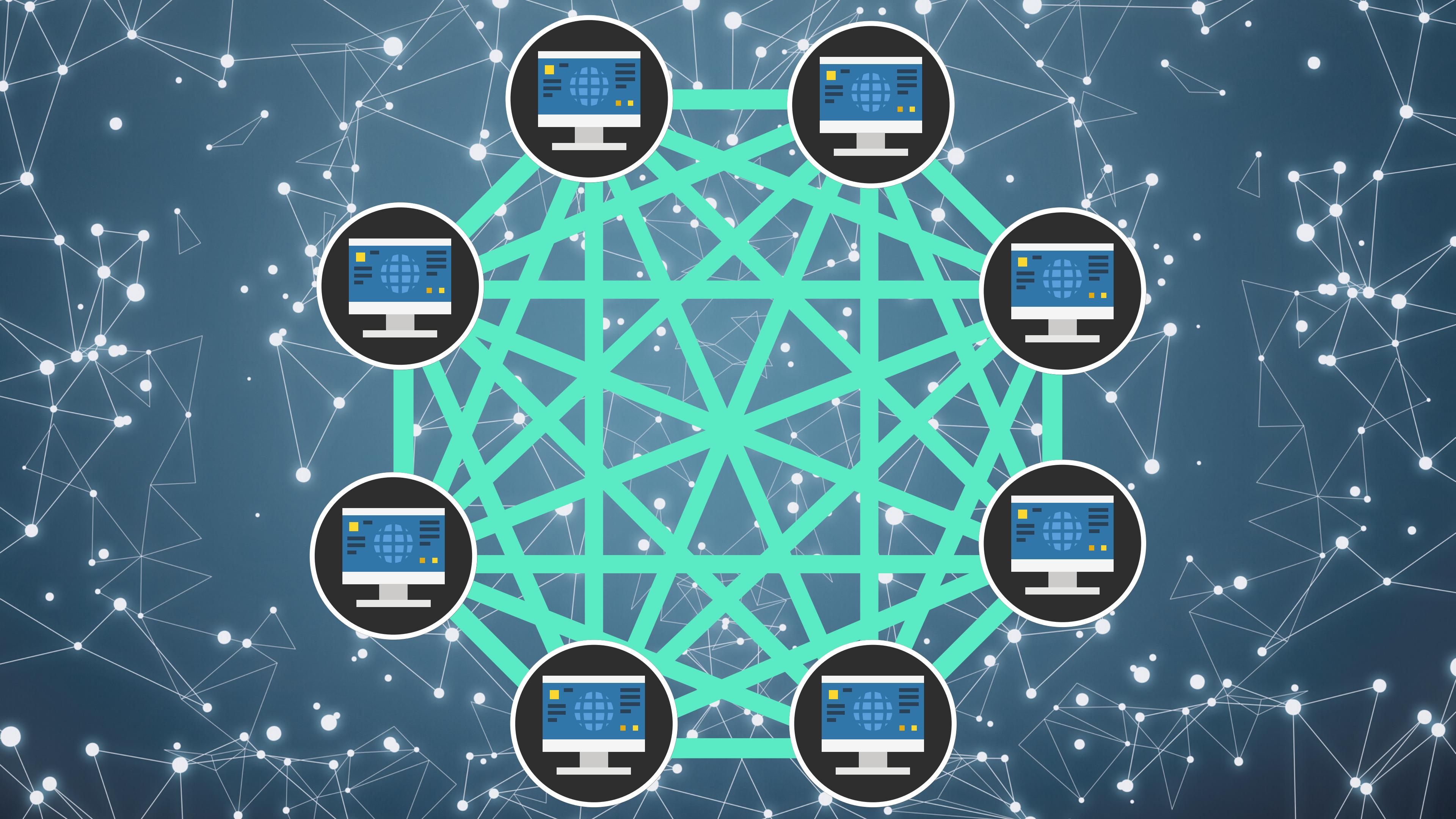
Internet?  
"A global network  
of networks."

(It's just a bunch of connected computers)

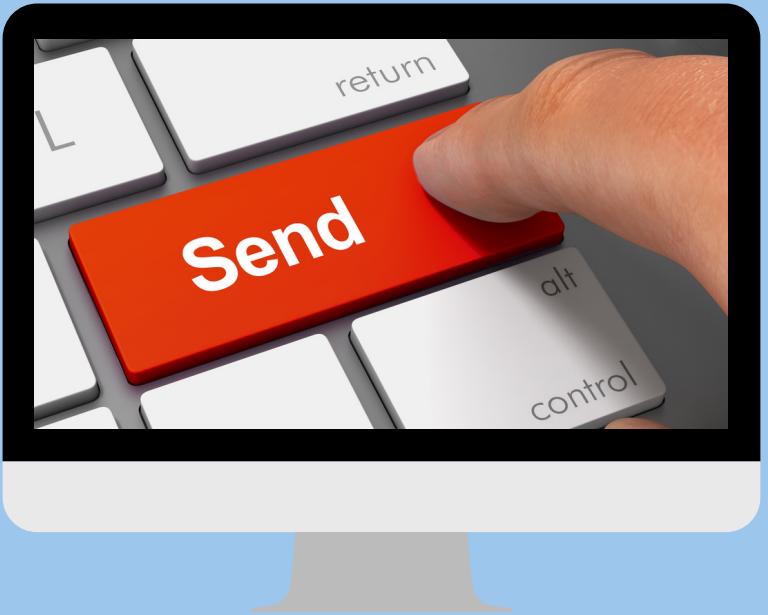


Hey

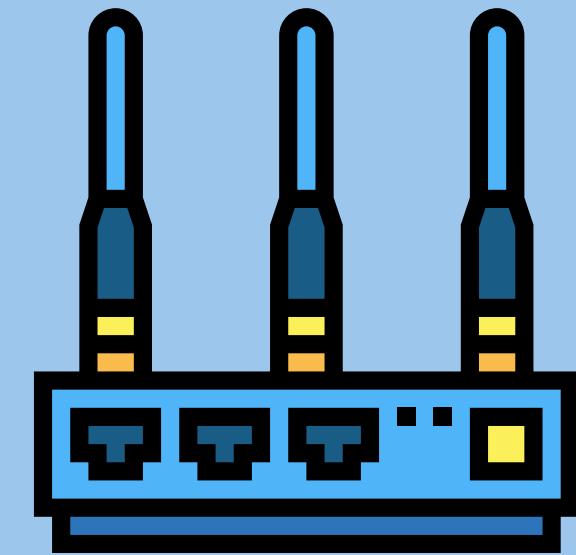




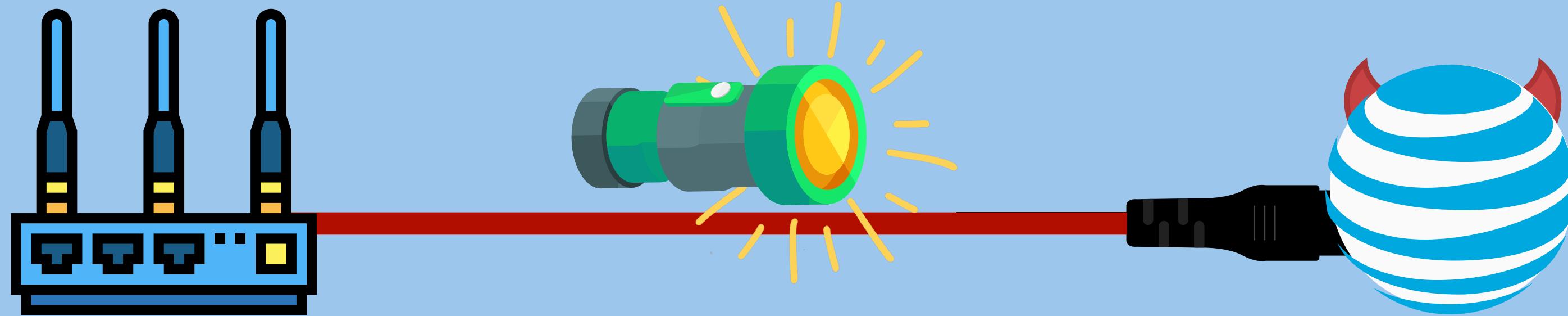




11010101010110101010  
01110101010101010111



1101010101011010100  
0111010101010101111



THE INTERNET IS THE INFRASTRUCTURE  
THAT CARRIES THINGS LIKE:

- EMAIL
- THE WEB
- FILE SHARING
- ONLINE GAMING
- STREAMING SERVICES

## THE INTERNET

Global network of interconnected computers that communicate via TCP/IP (don't worry about that for now). Network of networks.

## THE WEB

The World Wide Web is an information system where documents and other resources are available over the Internet.

Documents are transferred via **HTTP**.



# HTTP REQUESTS



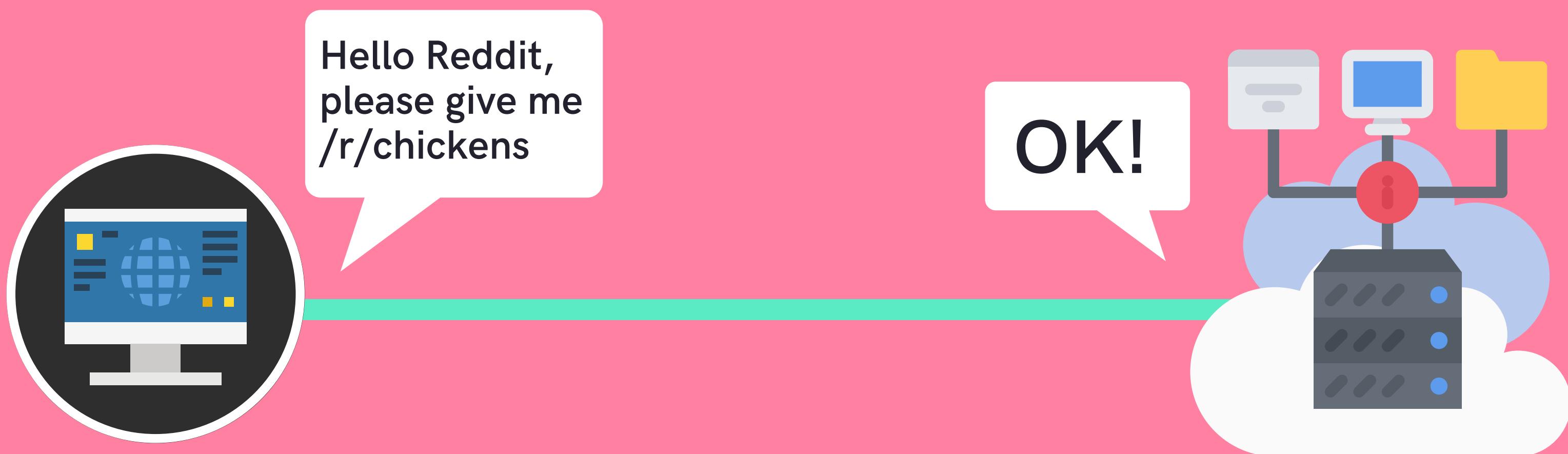
- Foundation of communication on the World Wide Web
- "Hyper Text Transfer Protocol"
- Request -> *I would like this information please*
- Response -> *Ok, here you go!*



# WEB SERVER

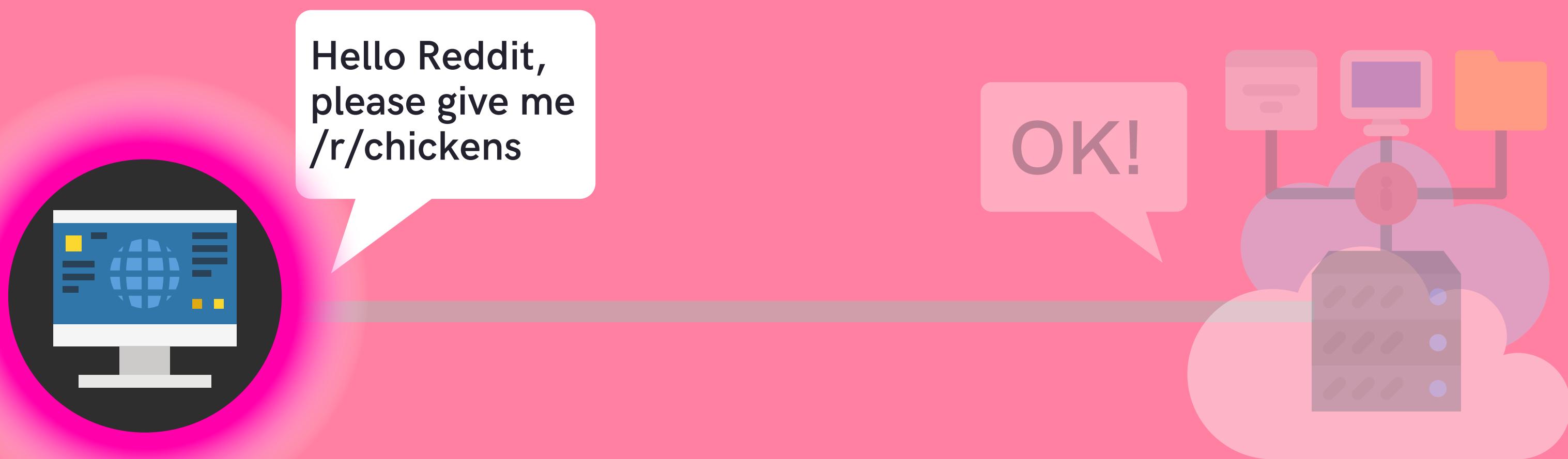
A computer\* that can satisfy requests on the Web.

\* "server" is also used to refer to the software running on the computer

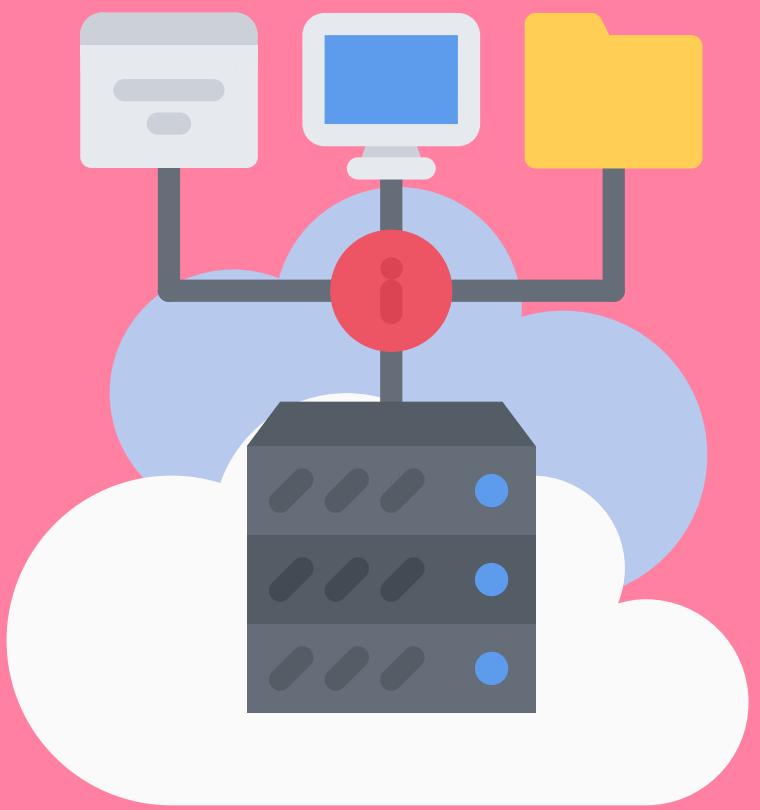


# CLIENT

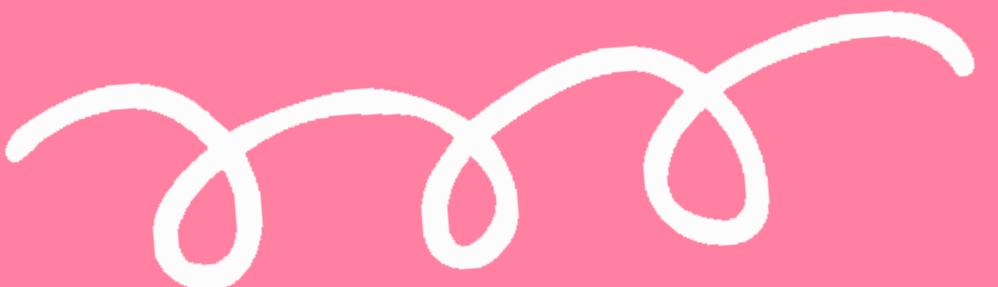
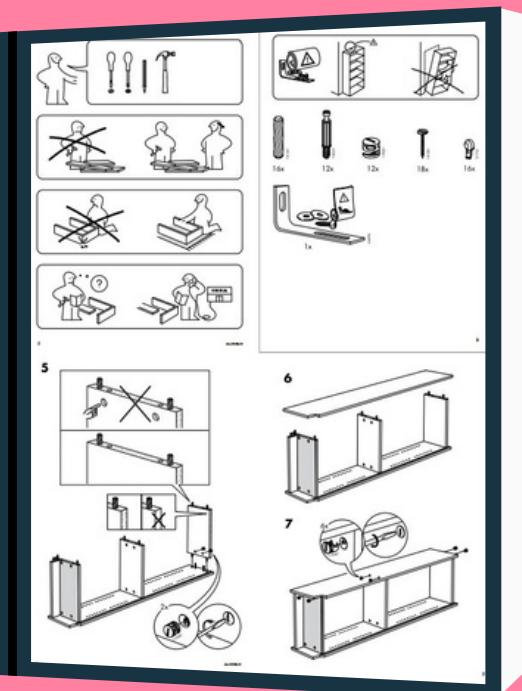
The computer that accesses a server



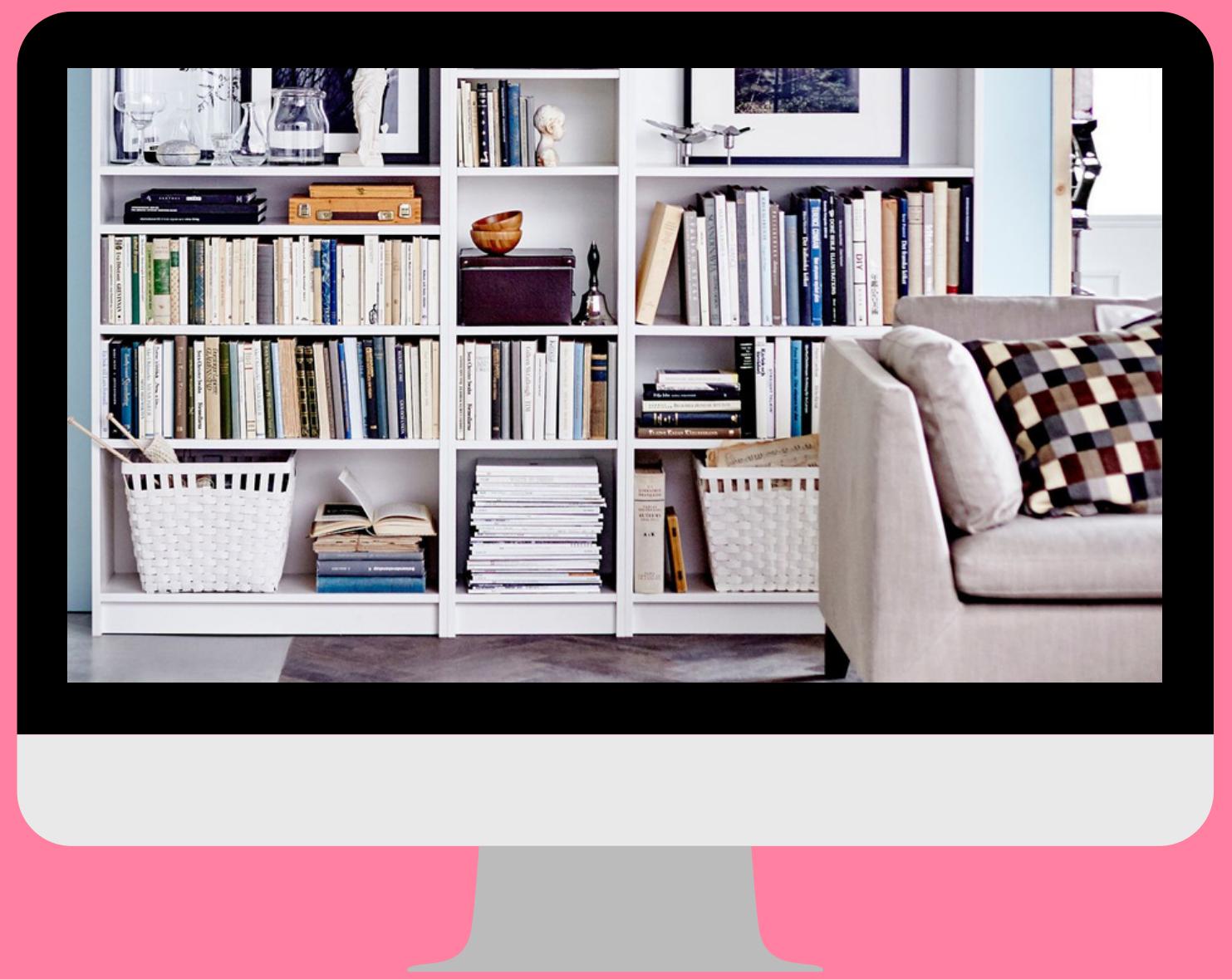
# A SERVER SOMEWHERE



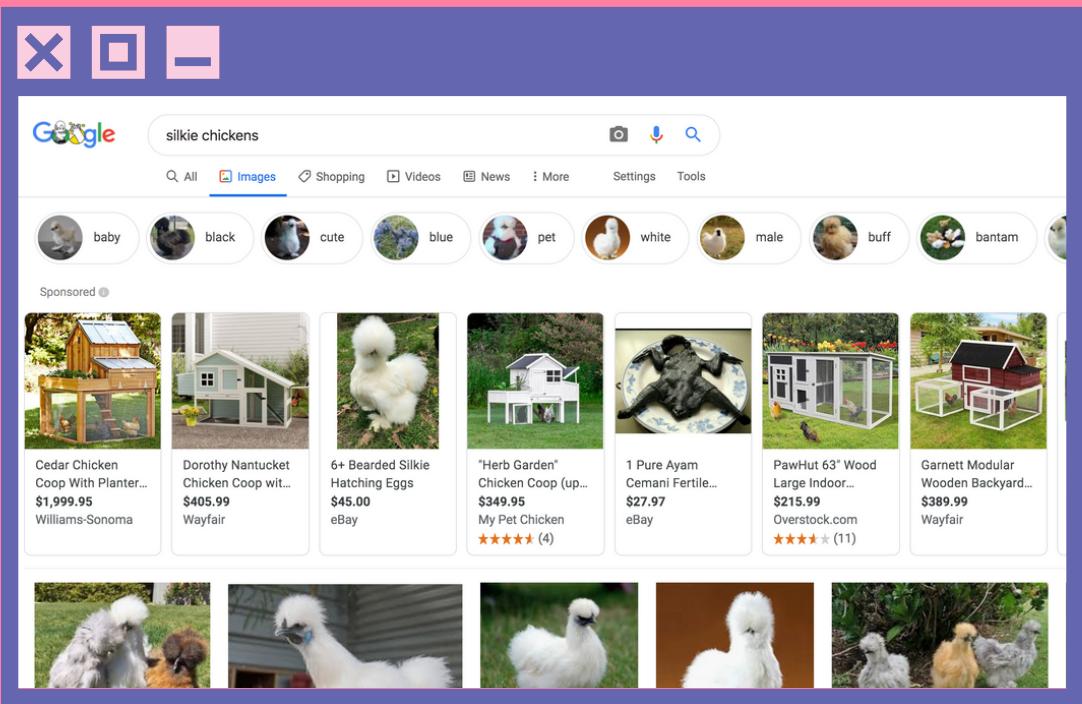
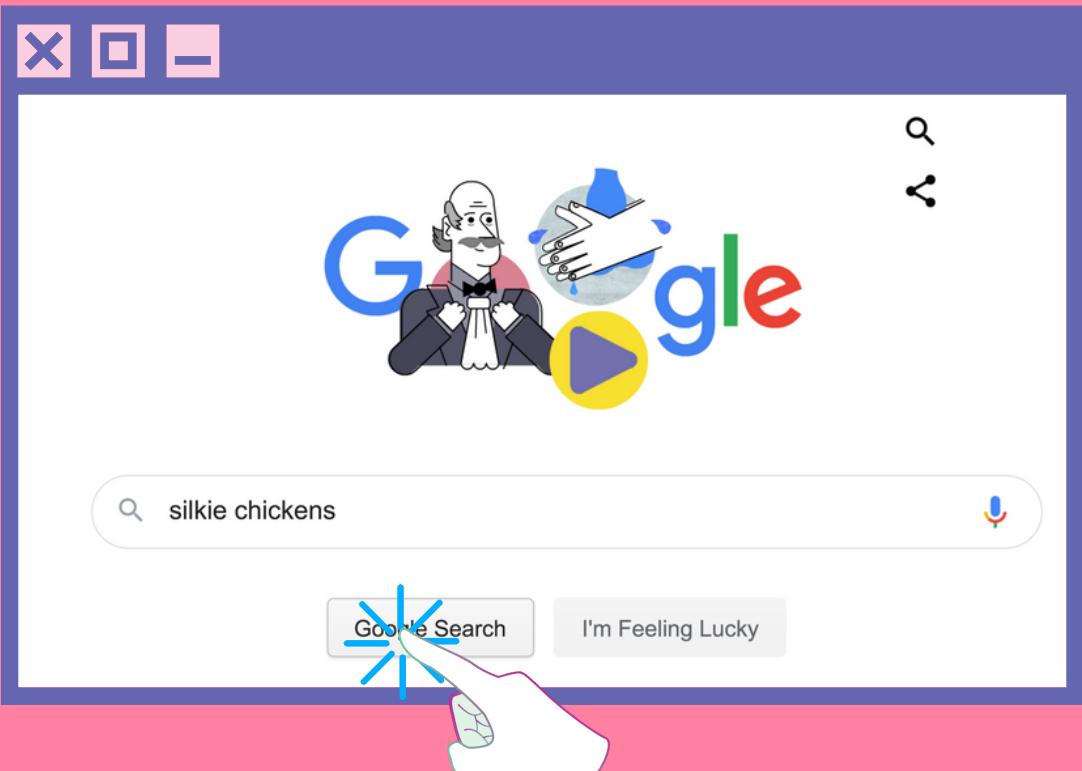
Here are the instructions!



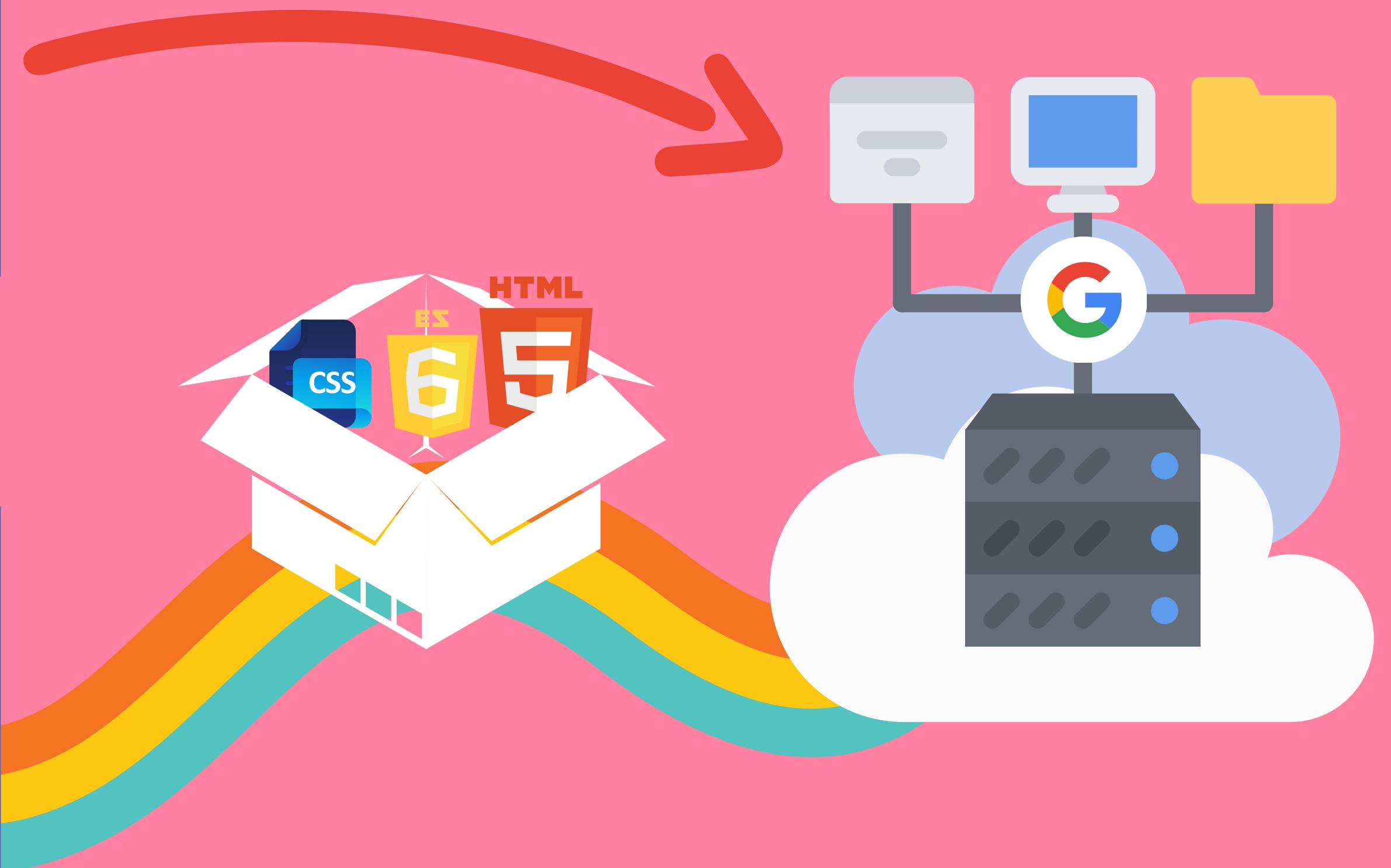
# YOUR BROWSER

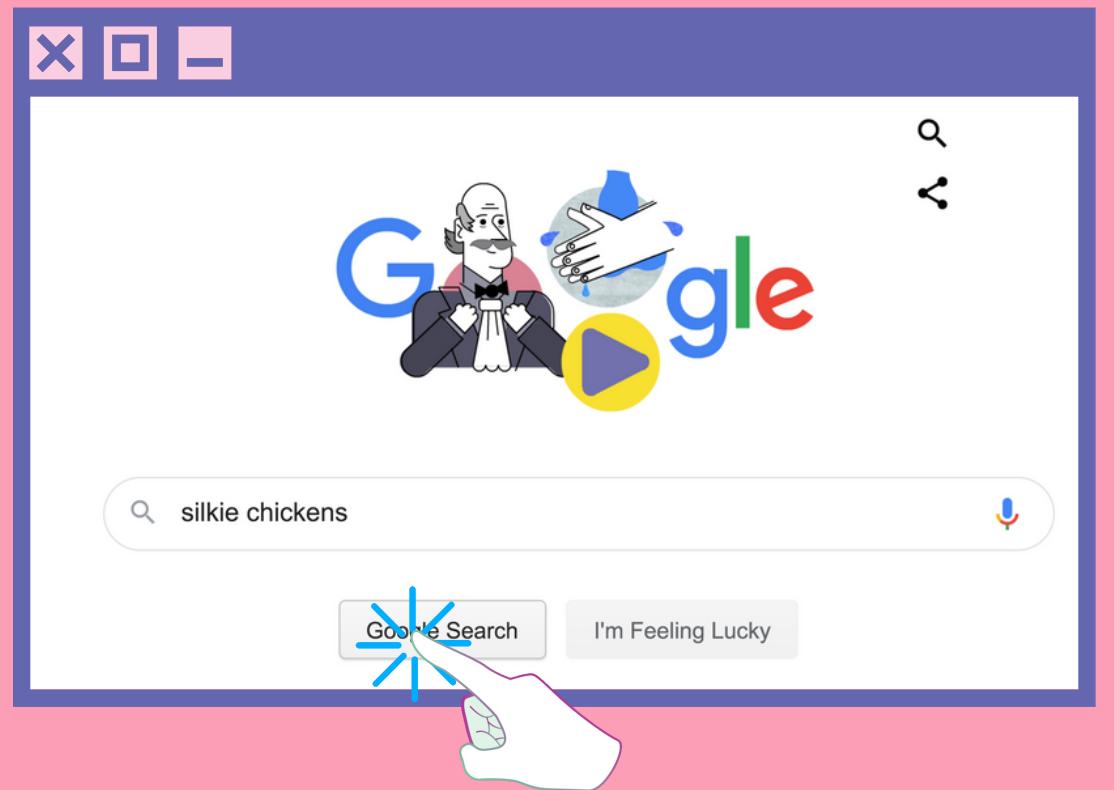


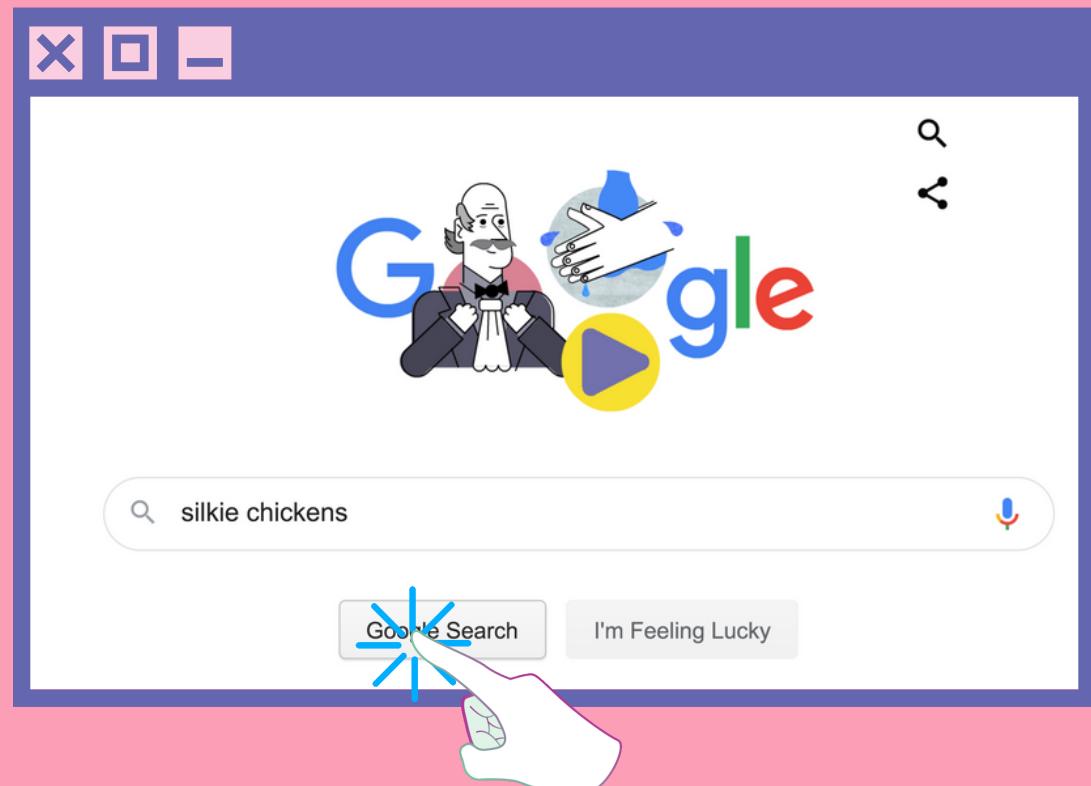
Look What I Made!



PLEASE GIVE ME  
GOOGLE.COM/SEARCH?Q=CHICKENS

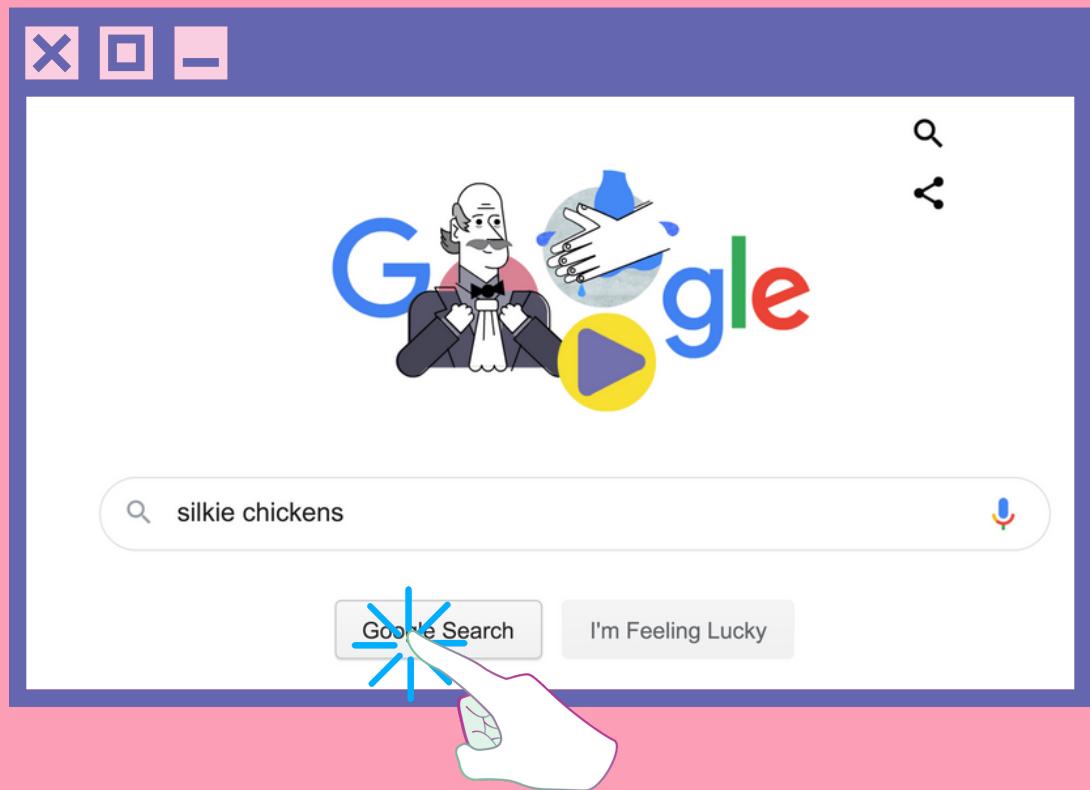




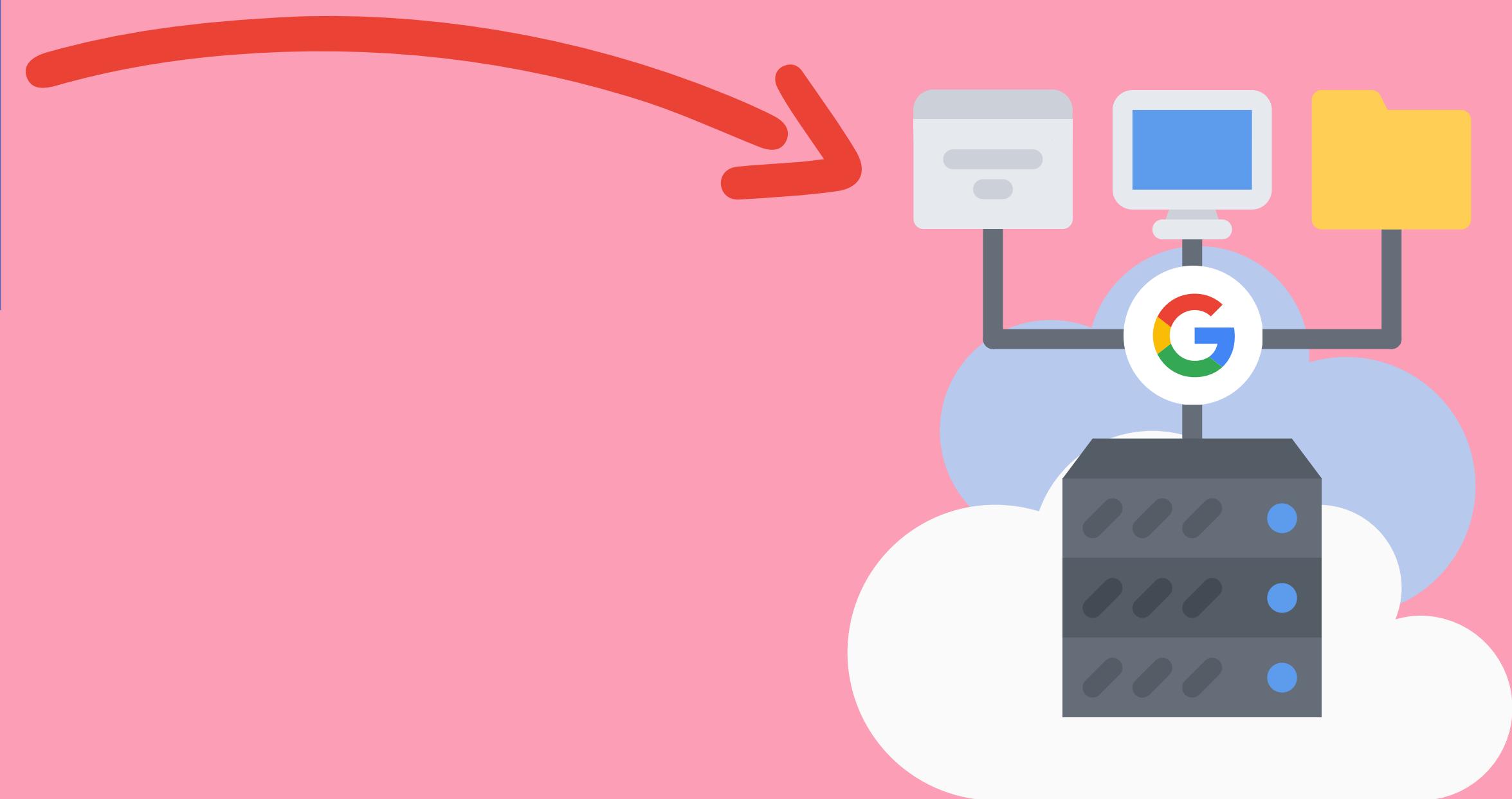


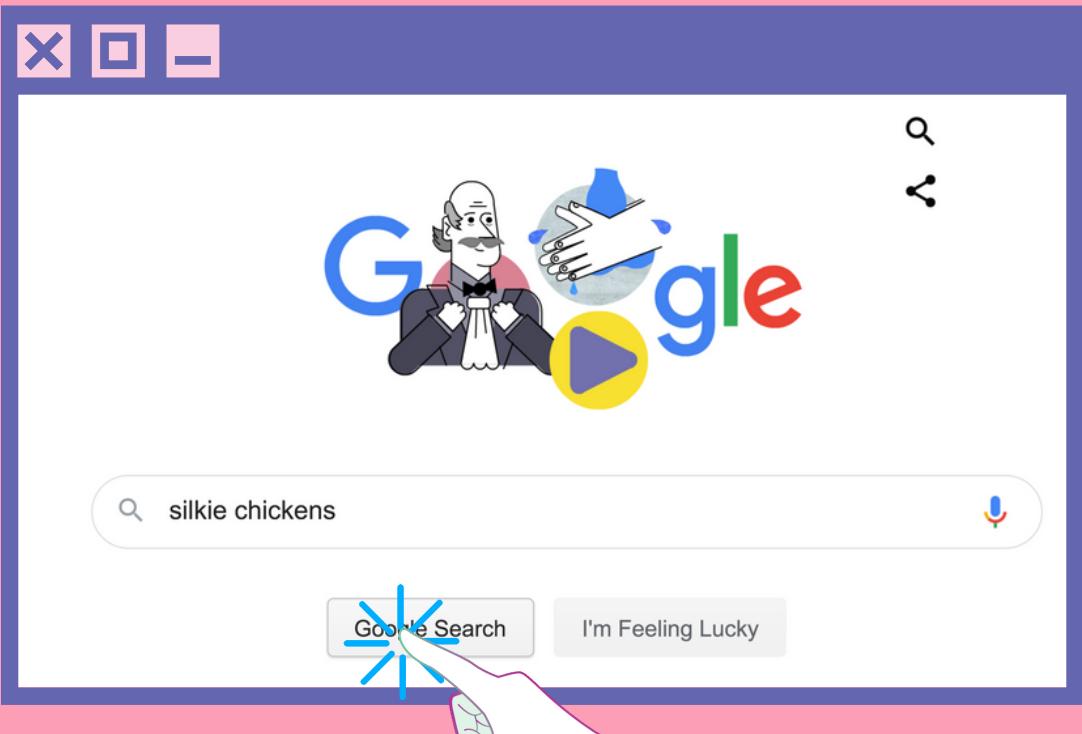
PLEASE GIVE ME  
GOOGLE.COM/SEARCH?Q=CHICKENS



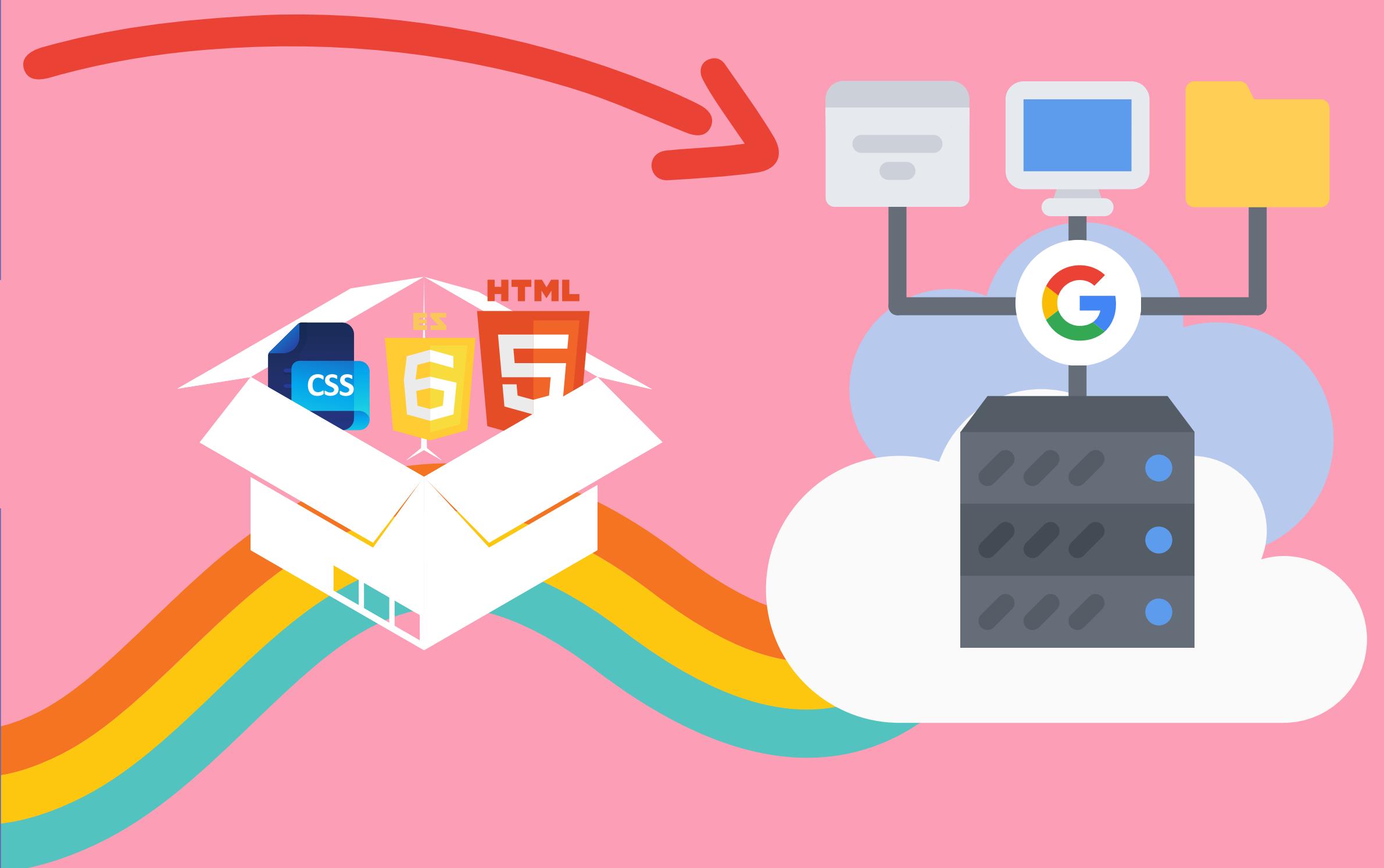
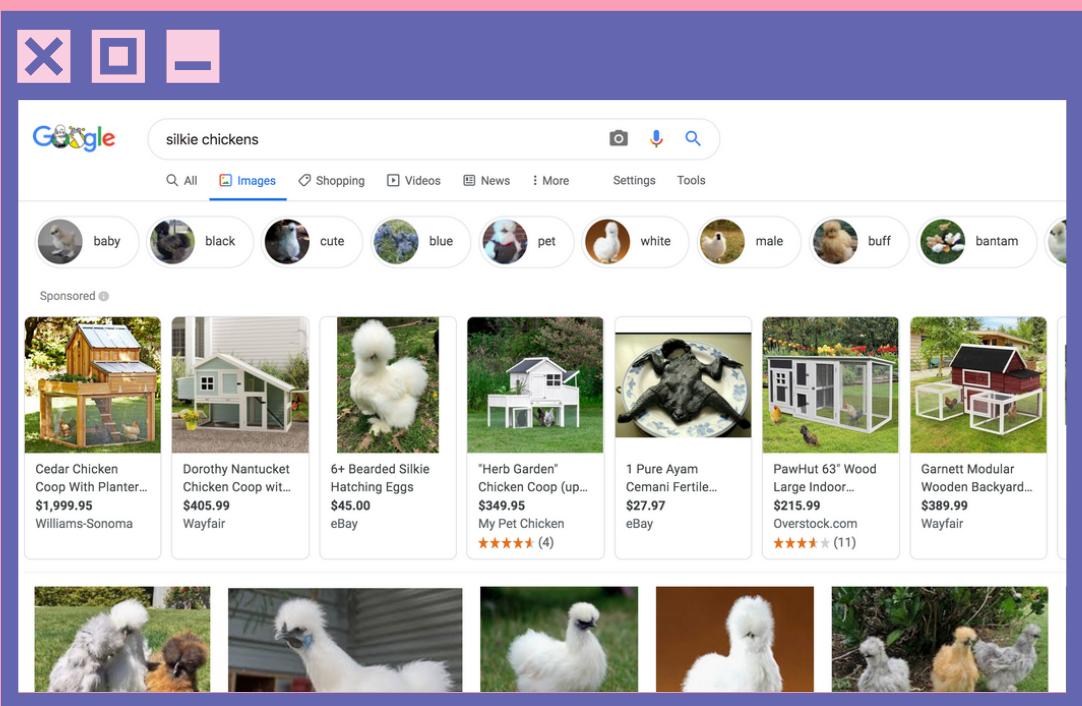


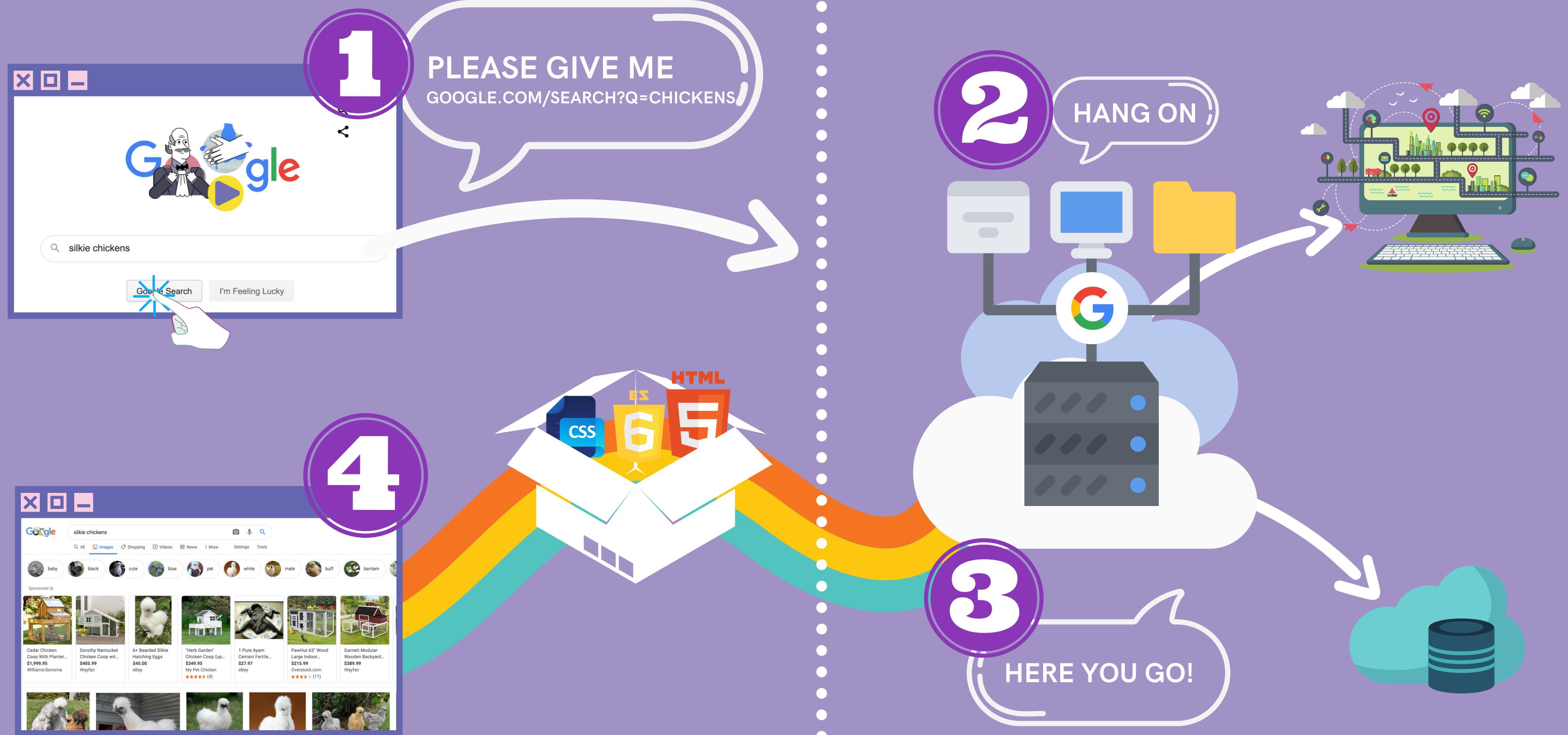
PLEASE GIVE ME  
GOOGLE.COM/SEARCH?Q=CHICKENS





PLEASE GIVE ME  
GOOGLE.COM/SEARCH?Q=CHICKENS





FRONT END

BACK END



FRONT END

BACK END

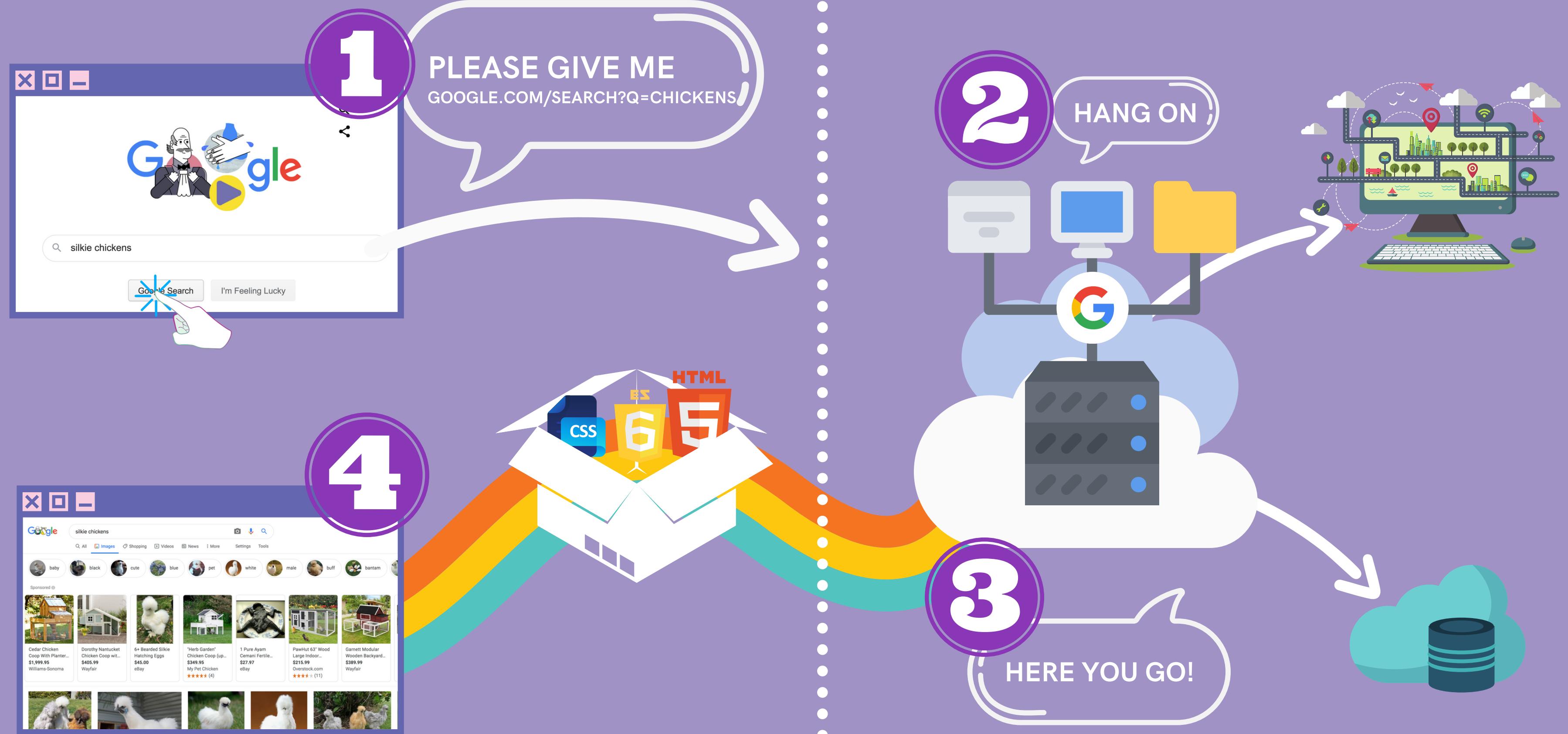


FRONT END



BACK END





FRONT END

BACK END

# HTML



# HTML IS A MARKUP LANGUAGE

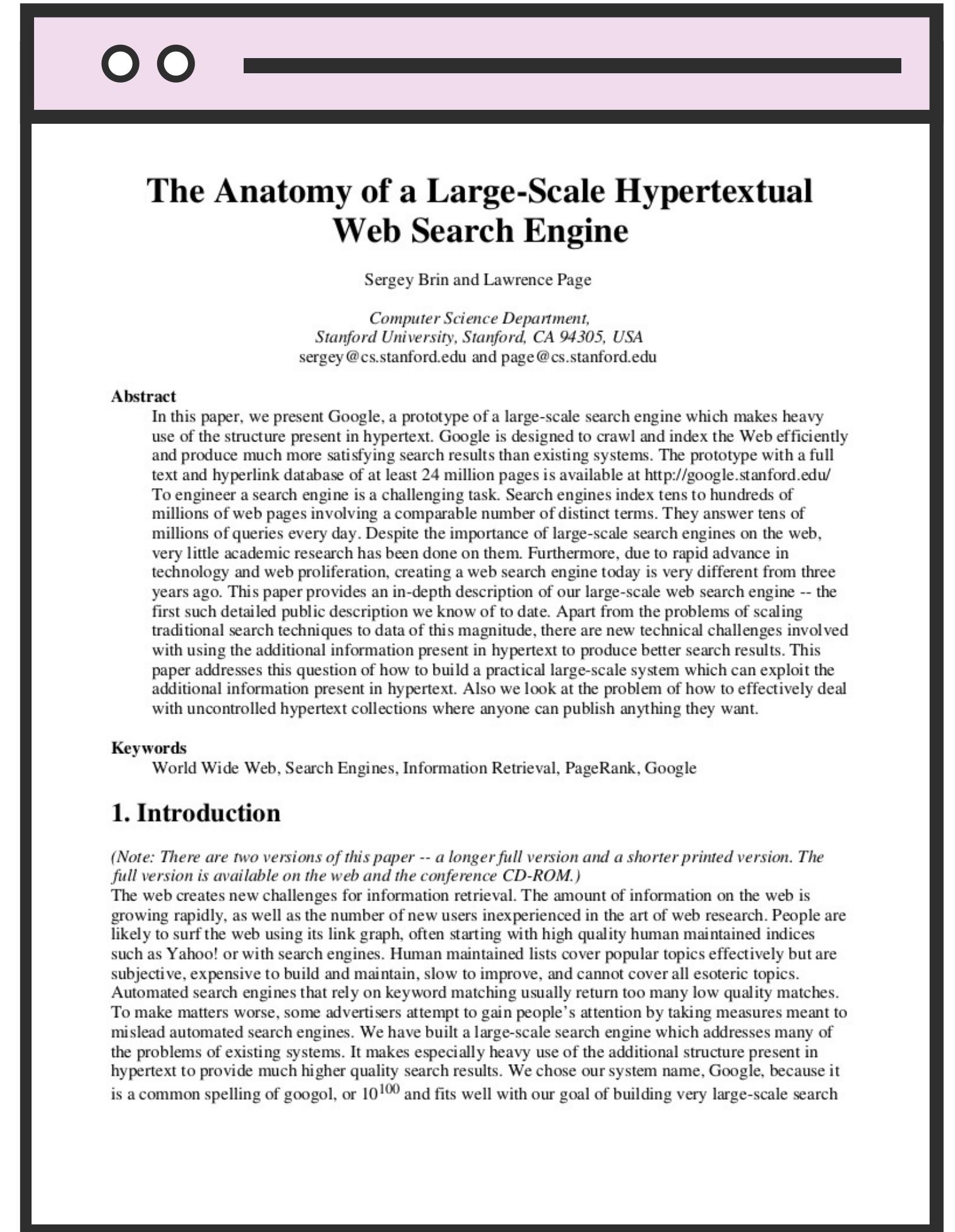
Colt, please see me after  
class. This is plagiarized.

To see a World in a Grain of  
Sand. And a Heaven in a Wild  
Flower Hold Infinity in the  
palm of your hand - And  
Eternity in an hour - A Robin  
Red breast in a Cage - Puts  
all Heaven in a Rage A Dove  
house *filld* with Doves &  
*Spelling* Pigeons

# MARKUP LANGUAGE

How would you describe this paper's structure to someone over the phone so that they could reproduce it?

What about morse code?

A screenshot of the original Google homepage from 1996. The page has a white background with a black header bar at the top. The title "The Anatomy of a Large-Scale Hypertextual Web Search Engine" is centered in a large, bold, black font. Below the title, the authors' names "Sergey Brin and Lawrence Page" are listed. Underneath their names is the text "Computer Science Department, Stanford University, Stanford, CA 94305, USA" and their email addresses "sergey@cs.stanford.edu" and "page@cs.stanford.edu". A section titled "Abstract" contains a detailed description of the paper's content. Another section titled "Keywords" lists "World Wide Web, Search Engines, Information Retrieval, PageRank, Google".

The Anatomy of a Large-Scale Hypertextual Web Search Engine

Sergey Brin and Lawrence Page

Computer Science Department,  
Stanford University, Stanford, CA 94305, USA  
sergey@cs.stanford.edu and page@cs.stanford.edu

**Abstract**

In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at <http://google.stanford.edu/>. To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been done on them. Furthermore, due to rapid advance in technology and web proliferation, creating a web search engine today is very different from three years ago. This paper provides an in-depth description of our large-scale web search engine -- the first such detailed public description we know of to date. Apart from the problems of scaling traditional search techniques to data of this magnitude, there are new technical challenges involved with using the additional information present in hypertext to produce better search results. This paper addresses this question of how to build a practical large-scale system which can exploit the additional information present in hypertext. Also we look at the problem of how to effectively deal with uncontrolled hypertext collections where anyone can publish anything they want.

**Keywords**

World Wide Web, Search Engines, Information Retrieval, PageRank, Google

## 1. Introduction

(Note: There are two versions of this paper -- a longer full version and a shorter printed version. The full version is available on the web and the conference CD-ROM.)

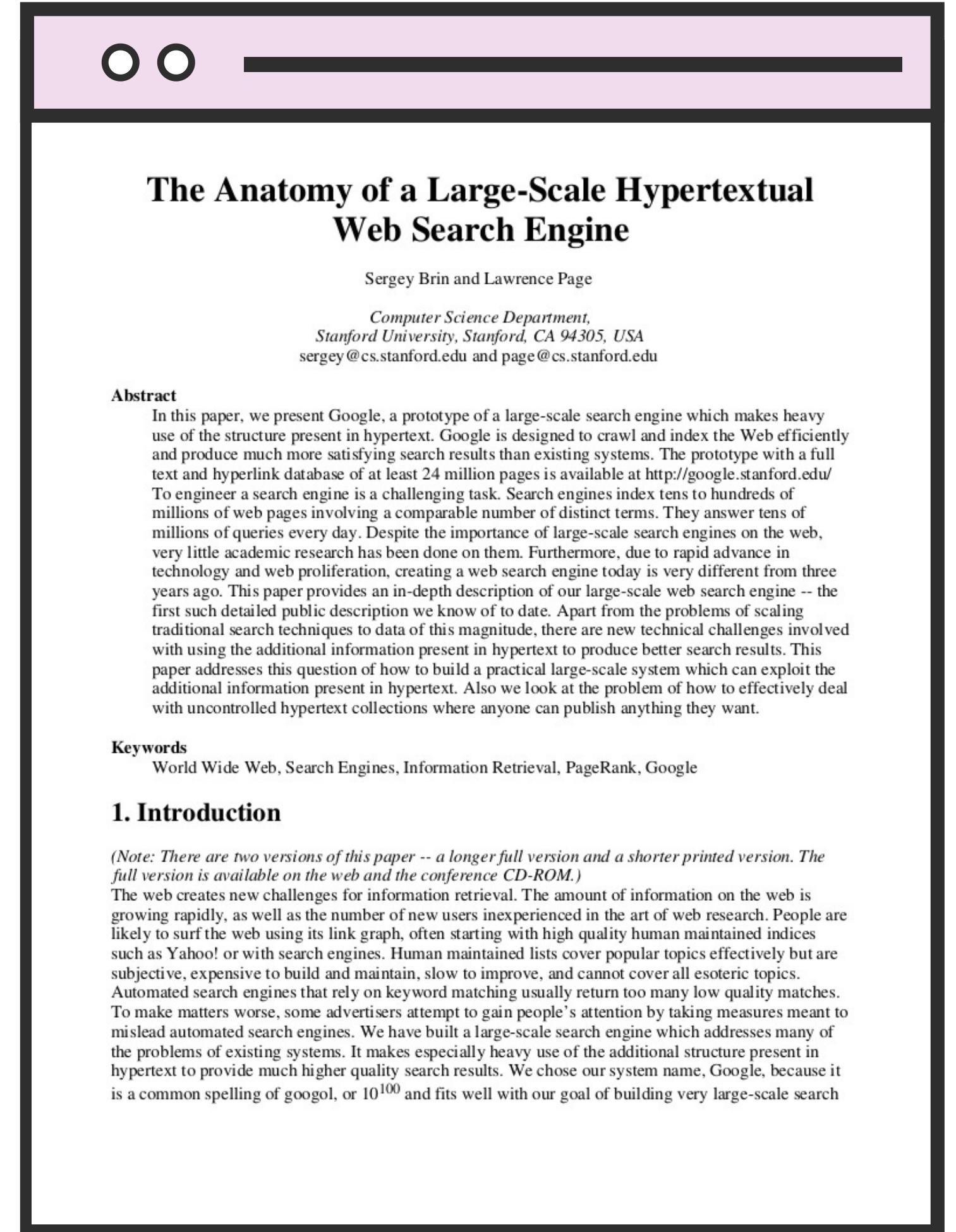
The web creates new challenges for information retrieval. The amount of information on the web is growing rapidly, as well as the number of new users inexperienced in the art of web research. People are likely to surf the web using its link graph, often starting with high quality human maintained indices such as Yahoo! or with search engines. Human maintained lists cover popular topics effectively but are subjective, expensive to build and maintain, slow to improve, and cannot cover all esoteric topics. Automated search engines that rely on keyword matching usually return too many low quality matches. To make matters worse, some advertisers attempt to gain people's attention by taking measures meant to mislead automated search engines. We have built a large-scale search engine which addresses many of the problems of existing systems. It makes especially heavy use of the additional structure present in hypertext to provide much higher quality search results. We chose our system name, Google, because it is a common spelling of googol, or  $10^{100}$  and fits well with our goal of building very large-scale search

# MARKUP LANGUAGE

"Make this part bold"

"Make this part a link"

"Make this a paragraph"



The screenshot shows the original Google homepage. At the top, there's a pink header bar with two circular icons on the left and a horizontal bar on the right. Below the header, the title "The Anatomy of a Large-Scale Hypertextual Web Search Engine" is displayed in a large, bold, black serif font. Underneath the title, the authors' names, "Sergey Brin and Lawrence Page", are listed in a smaller black font. Below their names, the text "Computer Science Department, Stanford University, Stanford, CA 94305, USA" and their email addresses, "sergey@cs.stanford.edu and page@cs.stanford.edu", are provided. A section titled "Abstract" begins with a detailed description of the Google prototype search engine, highlighting its efficiency and ability to handle millions of pages. The "Keywords" section includes "World Wide Web, Search Engines, Information Retrieval, PageRank, Google". Finally, the "1. Introduction" section starts with a note about the paper's availability and a detailed explanation of the challenges faced by search engines at the time.

**The Anatomy of a Large-Scale Hypertextual Web Search Engine**

Sergey Brin and Lawrence Page

Computer Science Department,  
Stanford University, Stanford, CA 94305, USA  
sergey@cs.stanford.edu and page@cs.stanford.edu

**Abstract**

In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at <http://google.stanford.edu/>. To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been done on them. Furthermore, due to rapid advance in technology and web proliferation, creating a web search engine today is very different from three years ago. This paper provides an in-depth description of our large-scale web search engine -- the first such detailed public description we know of to date. Apart from the problems of scaling traditional search techniques to data of this magnitude, there are new technical challenges involved with using the additional information present in hypertext to produce better search results. This paper addresses this question of how to build a practical large-scale system which can exploit the additional information present in hypertext. Also we look at the problem of how to effectively deal with uncontrolled hypertext collections where anyone can publish anything they want.

**Keywords**

World Wide Web, Search Engines, Information Retrieval, PageRank, Google

**1. Introduction**

(Note: There are two versions of this paper -- a longer full version and a shorter printed version. The full version is available on the web and the conference CD-ROM.)

The web creates new challenges for information retrieval. The amount of information on the web is growing rapidly, as well as the number of new users inexperienced in the art of web research. People are likely to surf the web using its link graph, often starting with high quality human maintained indices such as Yahoo! or with search engines. Human maintained lists cover popular topics effectively but are subjective, expensive to build and maintain, slow to improve, and cannot cover all esoteric topics. Automated search engines that rely on keyword matching usually return too many low quality matches. To make matters worse, some advertisers attempt to gain people's attention by taking measures meant to mislead automated search engines. We have built a large-scale search engine which addresses many of the problems of existing systems. It makes especially heavy use of the additional structure present in hypertext to provide much higher quality search results. We chose our system name, Google, because it is a common spelling of googol, or  $10^{100}$  and fits well with our goal of building very large-scale search



# HTML ELEMENTS

To write HTML, we pick from a set of standard Elements that all browsers recognize

Common Elements include:

- *<p> element* - represents a paragraph of text
- *<h1> element* - represents the main header on a page
- *<img> element* - embeds an image
- *<form> element* - represents a form

# HTML TAGS

We create elements by writing *tags*.

Most (but not all) elements consist of an opening and closing tag.

Opening Tag

```
<p>I am a paragraph</p>
```

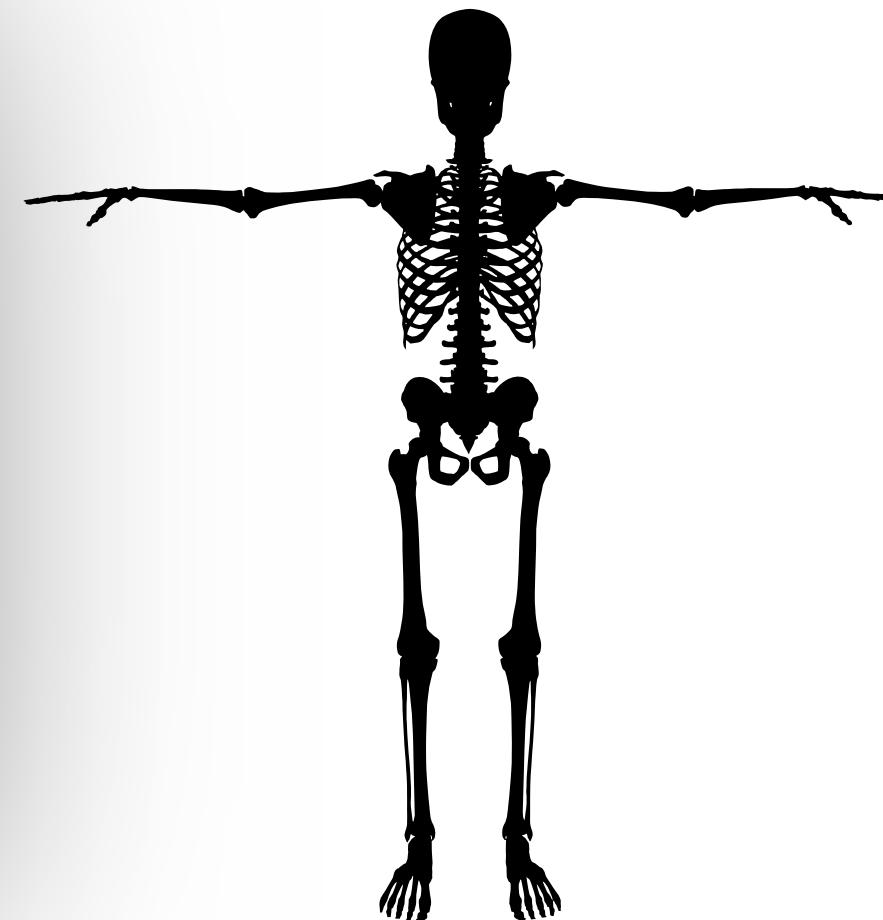
Closing Tag

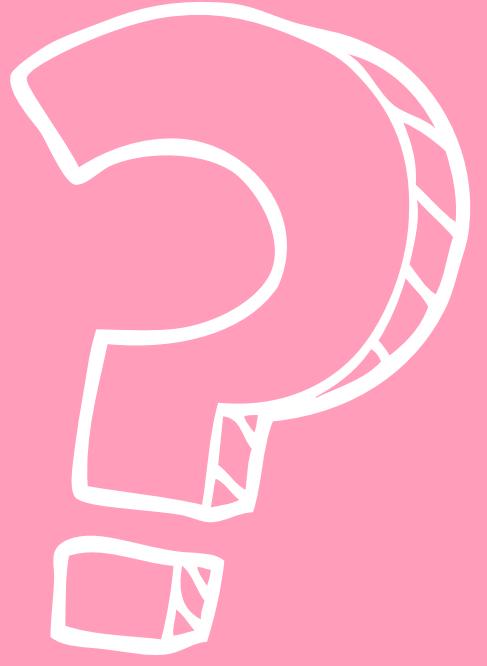


# HTML SKELETON

We write our HTML in a standard "skeleton"

```
● ● ●  
<!DOCTYPE html>  
<html>  
<head>  
  <title>My First Page</title>  
</head>  
<body>  
  <!-- Content Goes Here -->  
</body>  
</html>
```





## LIVING STANDARD

The HTML standard is a document that describes how HTML should work.

## ROLE OF BROWSERS

The standard describes the rules of HTML, but browsers actually have to do the work and implement HTML according to those rules.

# what is HTML5?

## HTML5?

HTML5 is the latest evolution of the standard that defines HTML. It includes new elements & features for browsers to implement,

# INLINE ELEMENTS



- Inline elements fit in alongside other elements
- Block level elements take up a whole "block" of space.

# BLOCK ELEMENTS





# SEMANTIC MARKUP

Semantic - relating to meaning

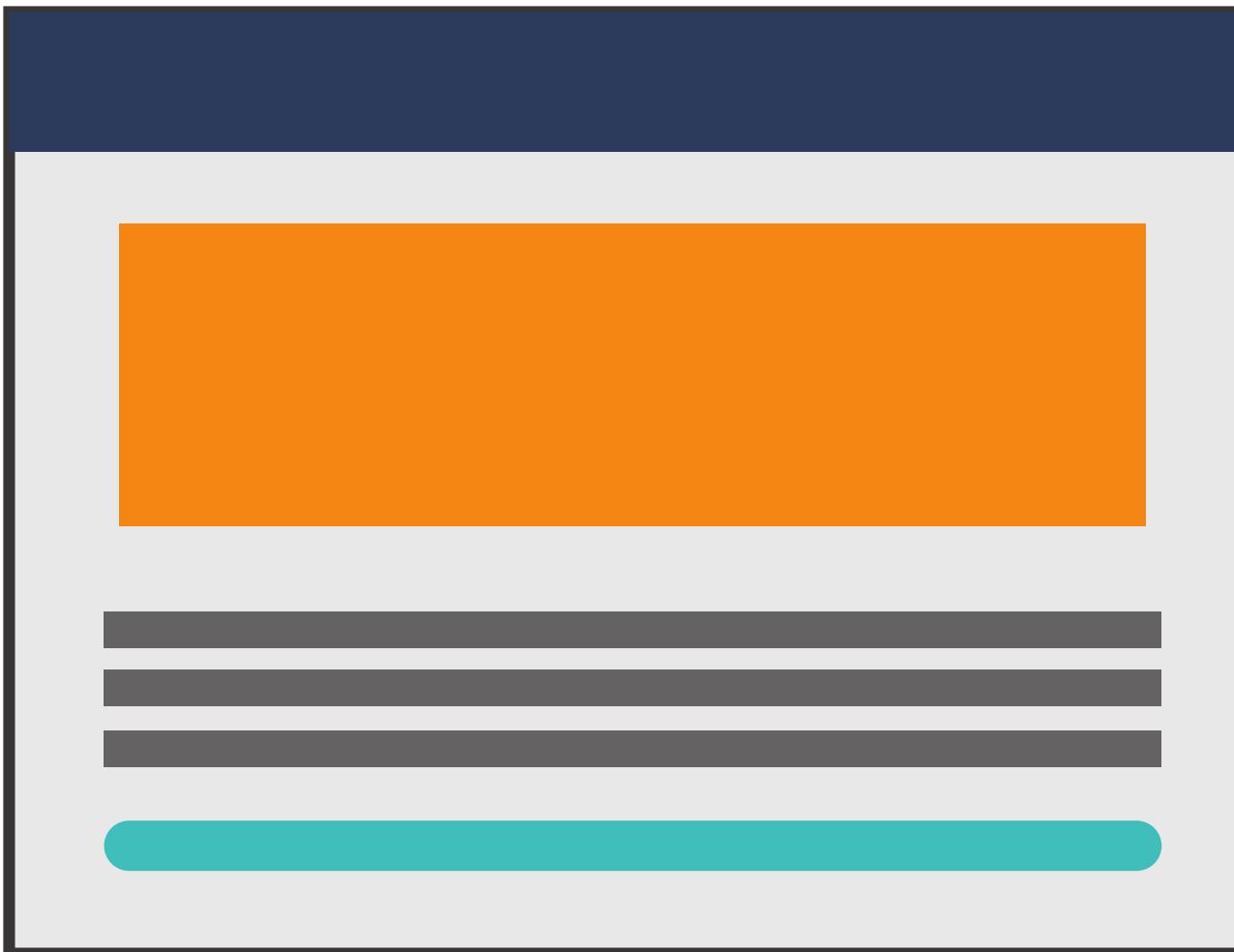
*"what purpose or role does that HTML element have?"*



# INSTEAD OF DIVS

Use more specific elements like:

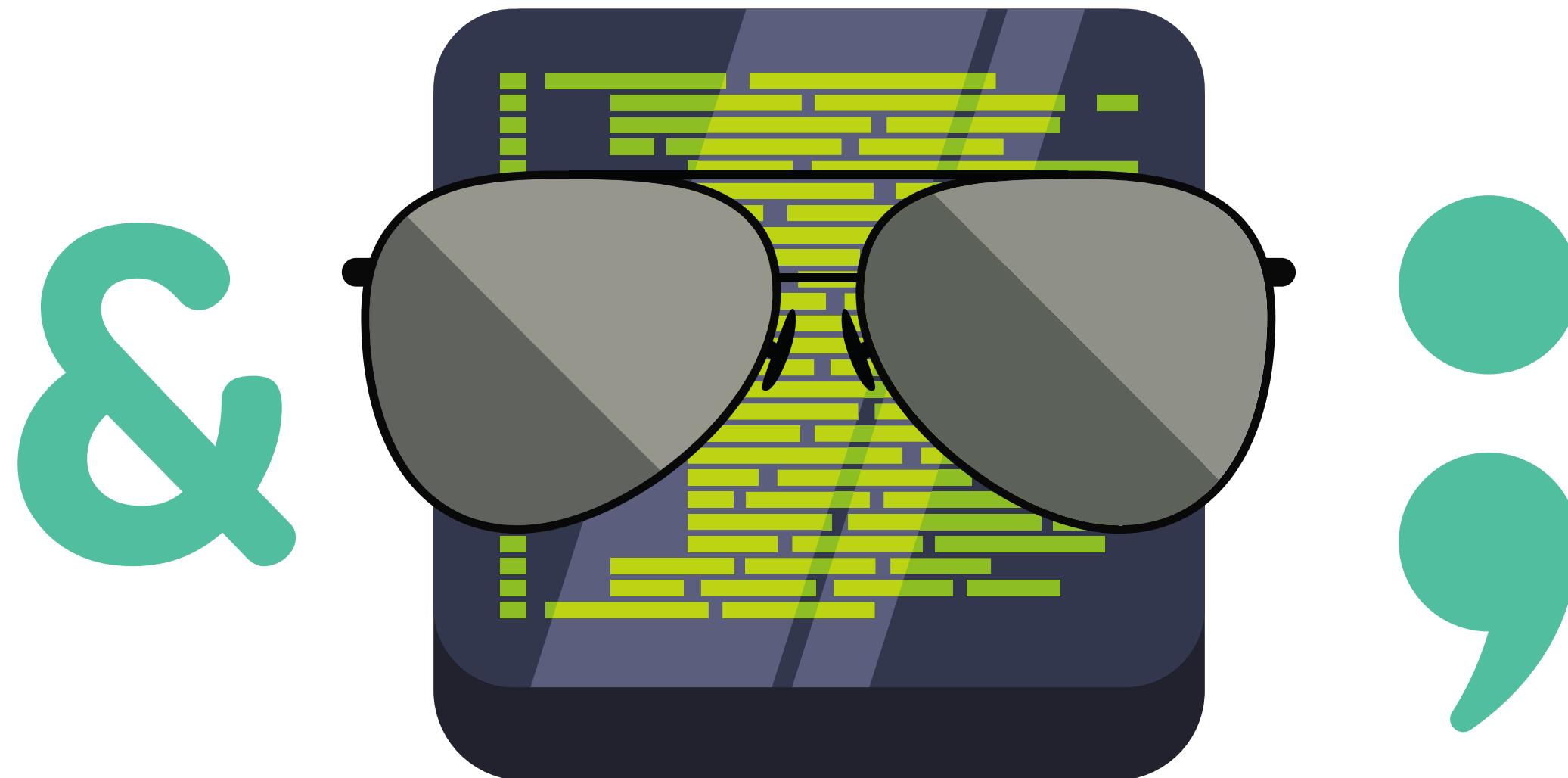
- <section>
- <article>
- <nav>
- <main>
- <header>
- <footer>
- <aside>
- <summary>





# HTML ENTITIES

WTF IS &#9824;



# HTML ENTITIES

---

- Start with an ampersand and end with a semicolon
- Used to display reserved characters, that normally would be invalid
- Also used in place of difficult to type characters
- The browser interprets them and renders the correct character instead.

# HTML Tables



-	-	-
-	-	-
-	-	-
-	-	-

## WHAT ARE TABLES?

Tables are structured sets of data, made up of rows and columns. They can be a great way of displaying data clearly.

## EARLY USAGE

In the early days of the web, tables were commonly used to create page layouts. Today, you should only use the table element when you are creating an actual data table.

# HTML TABLES

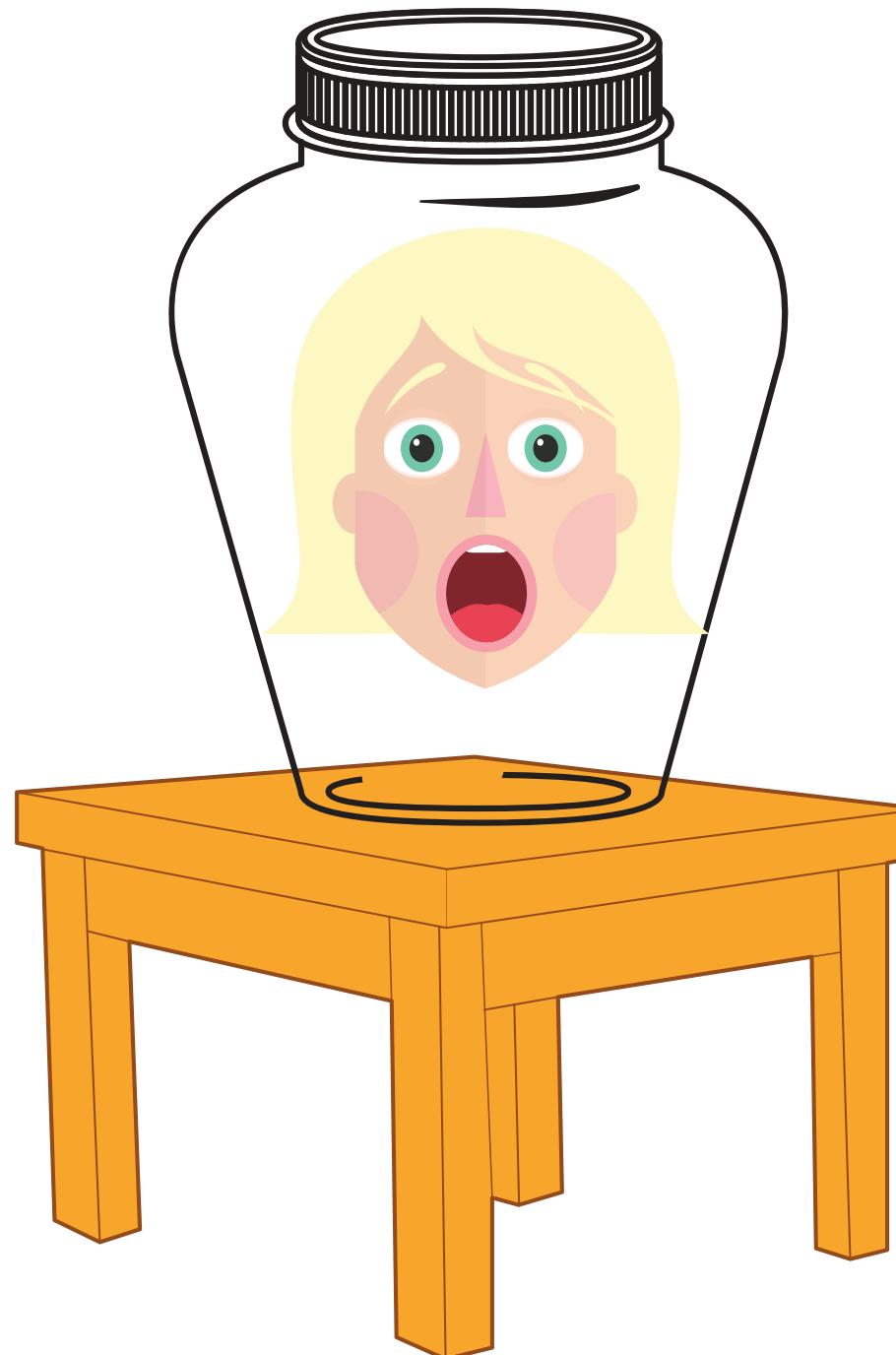
## LOTS OF ELEMENTS!

To create a table, you'll use 5-10 different elements! It can be tricky to remember them all, but don't panic!



# HTML TABLES

"WTF why are there so many elements just to make a table??"

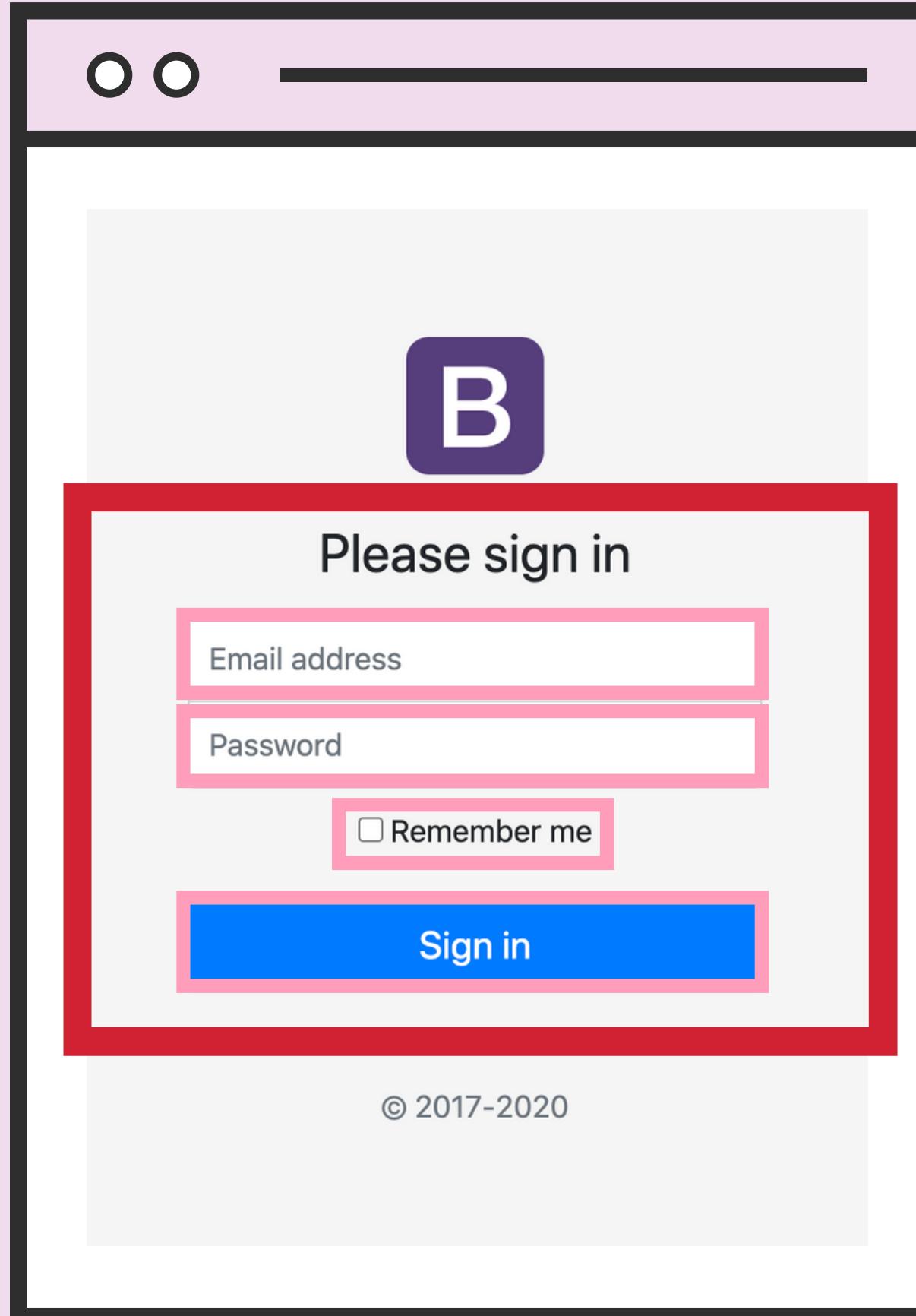
# ELEMENTS



<table>  
<td>  
<tr>  
<th>  
<thead>  
<tbody>  
<tfoot>  
<colgroup>  
<caption>

# HTML Forms





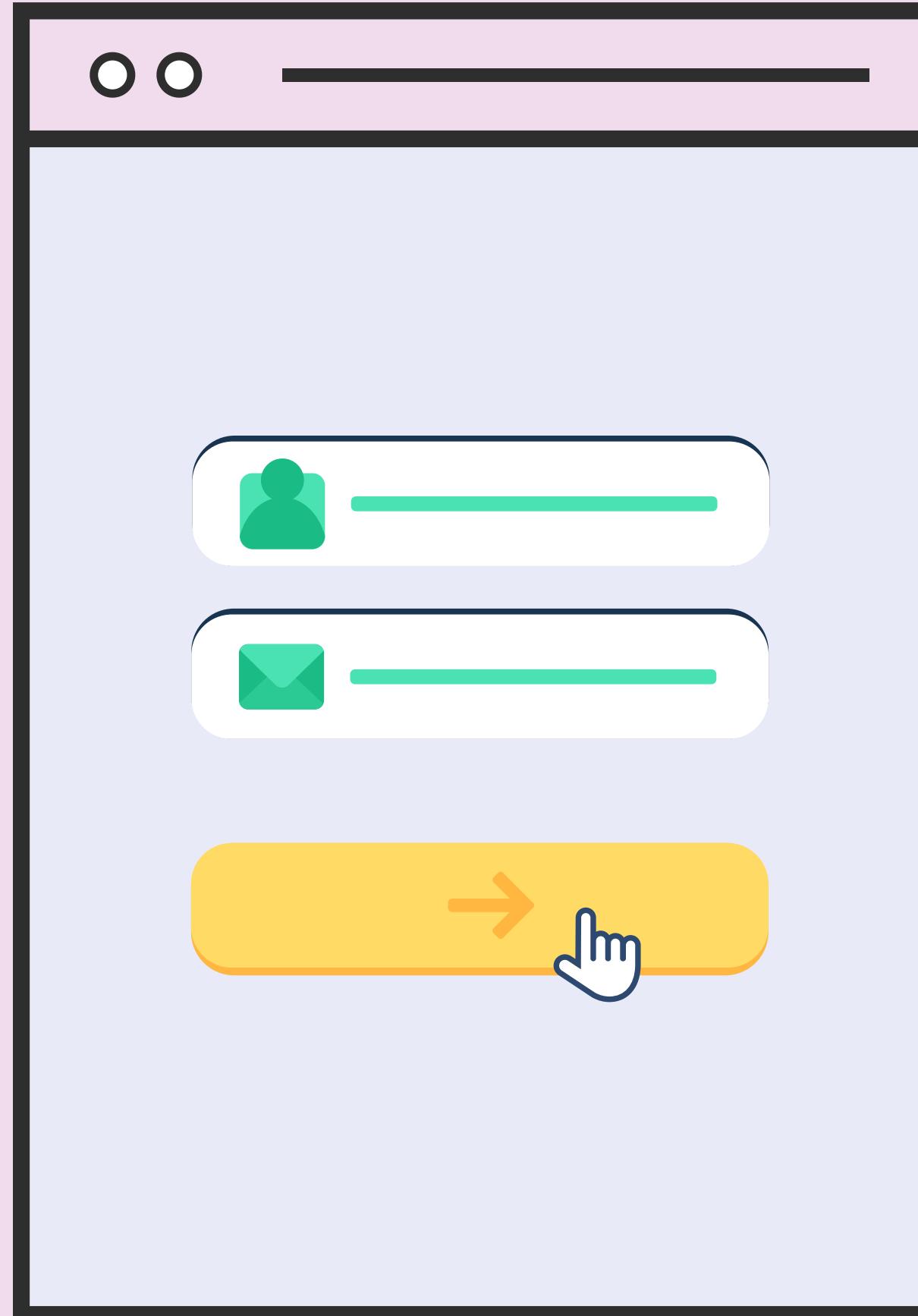
# CREATING FORMS

- The `<form>` element itself is a shell or container that doesn't have any visual impact.
- We then fill the form with a collection of inputs, checkboxes, buttons, etc.



# <form>

- The form element "represents a document section containing interactive controls for submitting information."
- The *action* attribute specifies WHERE the form data should be sent.
- The method attribute specifies which HTTP method should be used (don't worry about this for now)



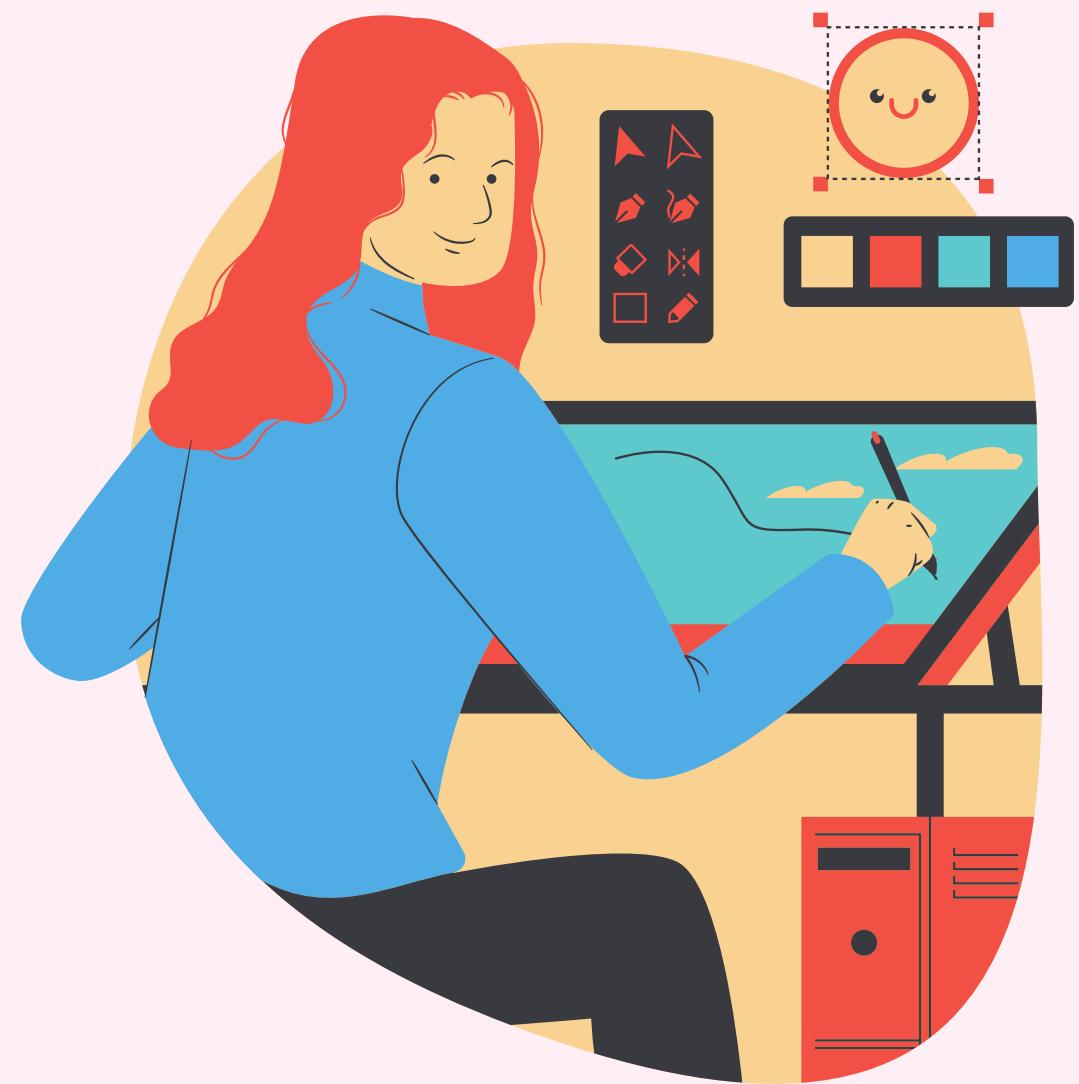
# <input>

- The `input` element is used to create a variety of different form controls.
- We have 20+ possible types of inputs, ranging from date pickers to checkboxes.
- The `type` attribute is where the magic happens. Changing `type` dramatically alters the input's behavior and appearance.



{ }

css



# CSS

## WHAT IS IT?

CSS is a language for describing how documents are presented visually - how they are arranged and styled.

## WHAT DOES IT STAND FOR?

CSS stands for Cascading Style Sheets. We'll cover the "cascading" part in a bit; don't worry about it for now!

## THERE'S A LOT!

CSS is very easy to get the hang of, but it can be intimidating because of how many properties we can manipulate.

# CSS RULES

(almost) everything you do in CSS follows this basic pattern:

```
selector {  
    property: value;  
}
```

# CSS RULES

Make all `<h1>` elements purple

```
h1 {  
    color: purple;  
}
```

# CSS RULES

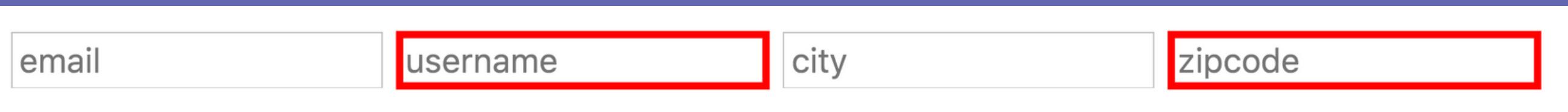
Make all image elements  
100 pixels wide & 200 pixels tall

```
img {  
    width: 100px;  
    height: 200px;  
}
```

# FANCIER!

Select every other text input  
and give it a red border:

```
input[type="text"]:nth-of-type(2n){  
    border:2px solid red;  
}
```

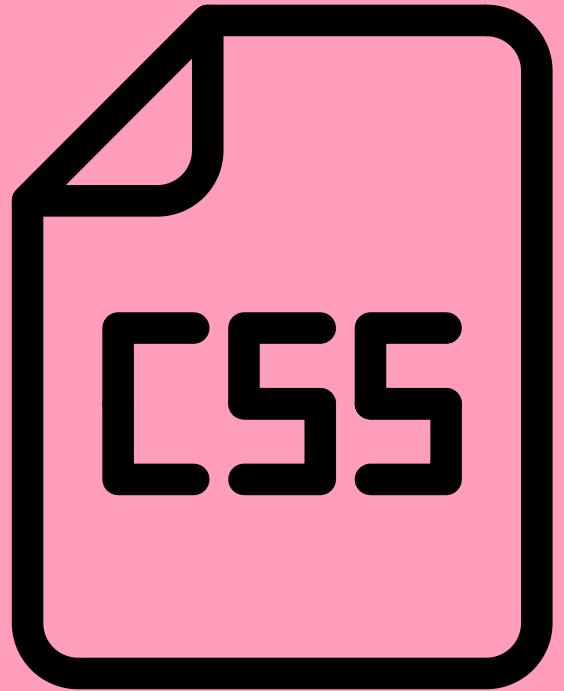


email      username      city      zipcode

# So Many CSS Border Properties



border border-blockborder-block-color border-block-end border-block-end-color border-block-end-style border-block-end-width border-block-start border-block-start-color border-block-start-style border-block-start-width border-block-style border-block-width border-bottom border-bottom-color border-bottom-left-radius border-bottom-right-radius border-bottom-style border-bottom width border-collapse border-color border-end-end-radiusborder-end-start-radiusborder-imageborder-image-outsetborder-image-repeatborder-image-sliceborder-image-sourceborder-image-widthborder-inlineborder-inline-colorborder-inline-endborder-inline-end-colorborder-inline-end-styleborder-inline-end-widthborder-inline-startborder-inline-start-colorborder-inline-start-styleborder-inline-start-widthborder-inline-styleborder-inline-widthborder-leftborder-left-colorborder-left-styleborder-left-widthborder-radiusborder-rightborder-right-colorborder-right-styleborder-right-widthborder-spacingborder-start-end-radiusborder-start-start-radiusborder-styleborder-topborder-top-colorborder-top-left-radiusborder-top-right-radiusborder-top-styleborder-top-widthborder-width



# Including Styles

## INLINE STYLES

You can write your styles directly inline on each element, but this is **NOT A GOOD IDEA** most of the time.

## THE <STYLE> ELEMENT

You can write your styles inside of a `<style>` element. This is easy, but it makes it impossible to share styles between documents. **Not recommended either.**

## EXTERNAL STYLESHEET

Write your styles in a `.css` file, and then include the using a `<link>` in the head of your html document. **Recommended!**

# <link>



```
<head>
  <title>Forms Demo</title>
  <link rel="stylesheet" href="my_styles.css">
</head>
```



# css colors



**NAMED  
COLORS**

**Hotpink**

**Mediumorchid**

**Firebrick**

**Darkkhaki**

**Gold**

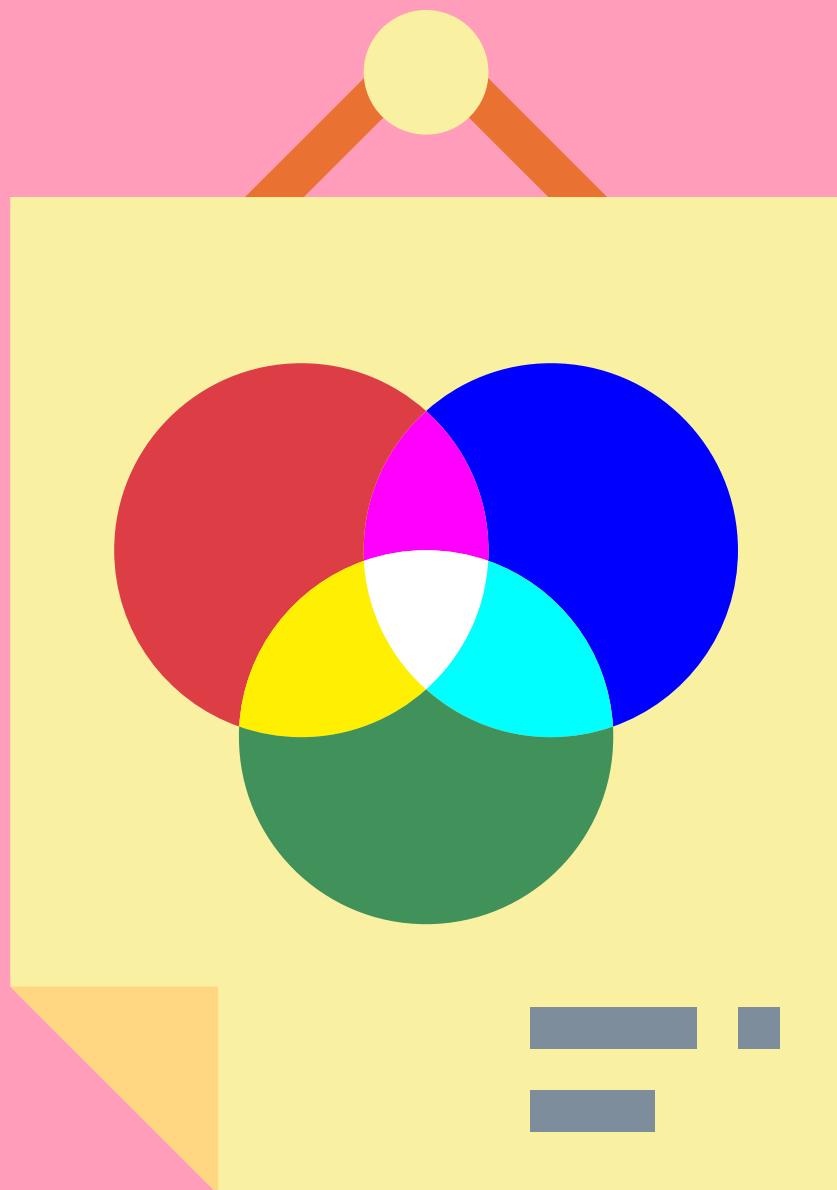
**MediumAquamarine**

**Lightskyblue**

**Tomato**

A typical computer can display  
~16,000,000  
different colors





# RGB

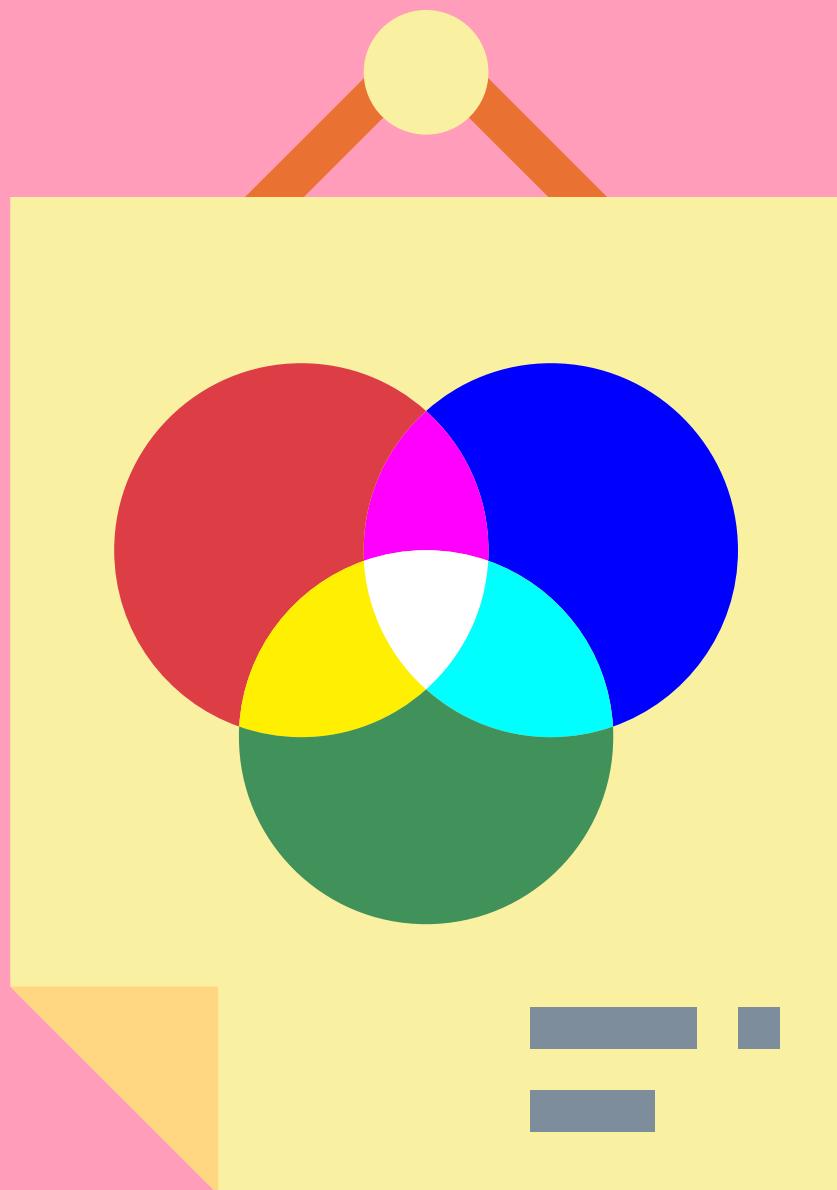
- Red, Green, Blue Channels
- Each channel ranges from 0-255

**rgb(255,0,0)**

**rgb(0,0,255)**

**rgb(173, 20, 219)**

**rgb(0,0,0)**



# Hex

- Still red, green, and blue channels
- Each ranges from 0-255 BUT represented with hexadecimal

# Decimal

0,1,2,3,4,

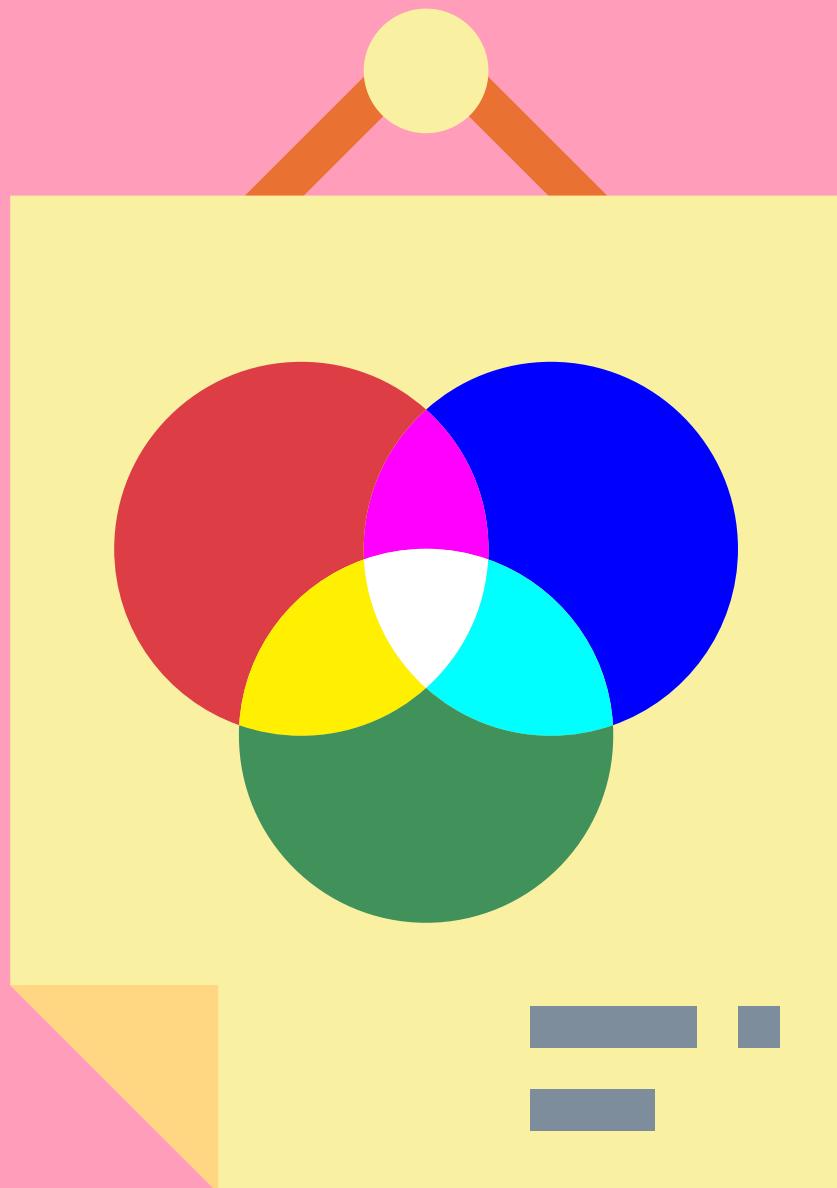
5,6,7,8,9

# Hexadecimal

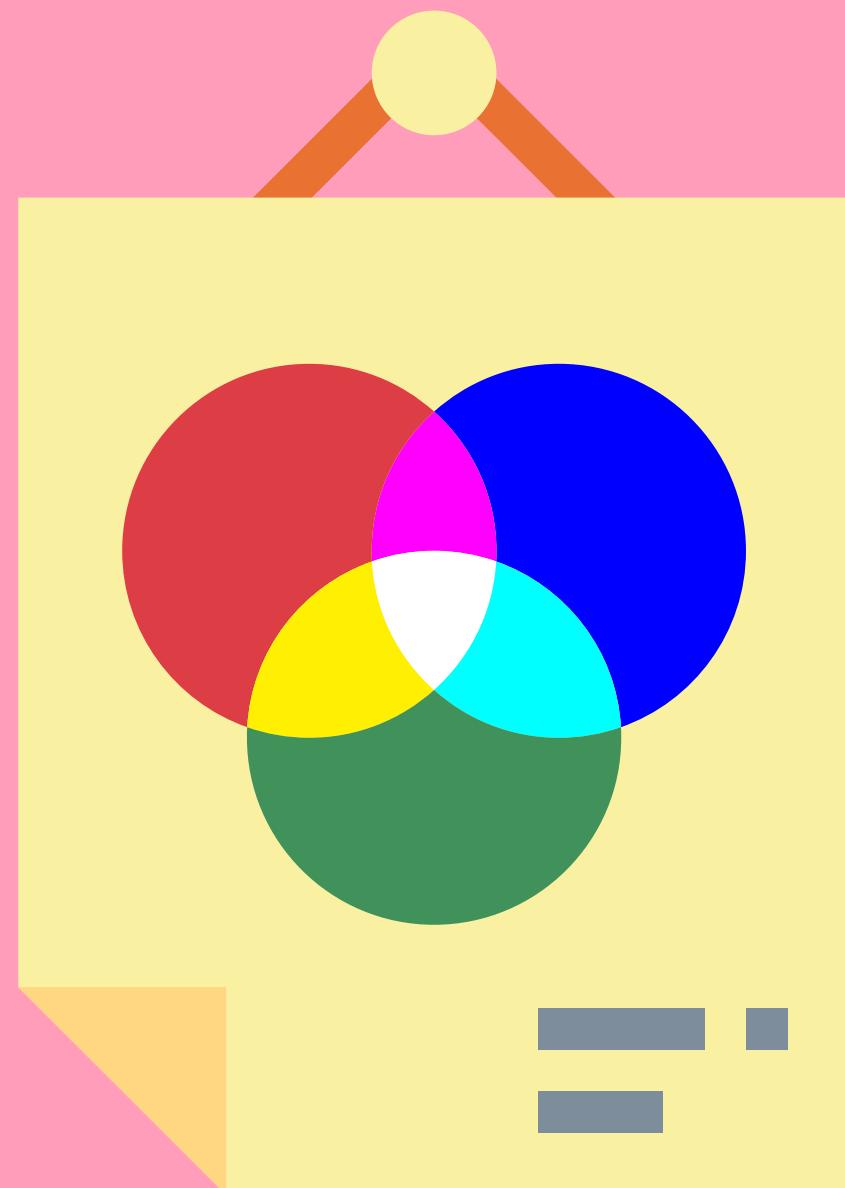
0, 1, 2, 3, 4,

5, 6, 7, 8, 9,

A, B, C, D, E, F



#fffff00  
red    green    blue



#0f5679

red

green

blue



# CSS Text Properties

- **text-align**
- **font-weight**
- **text-decoration**
- **line-height**
- **letter-spacing**



**FONT  
SIZE**





## Relative

- EM
- REM
- VH
- VW
- %
- AND MORE!

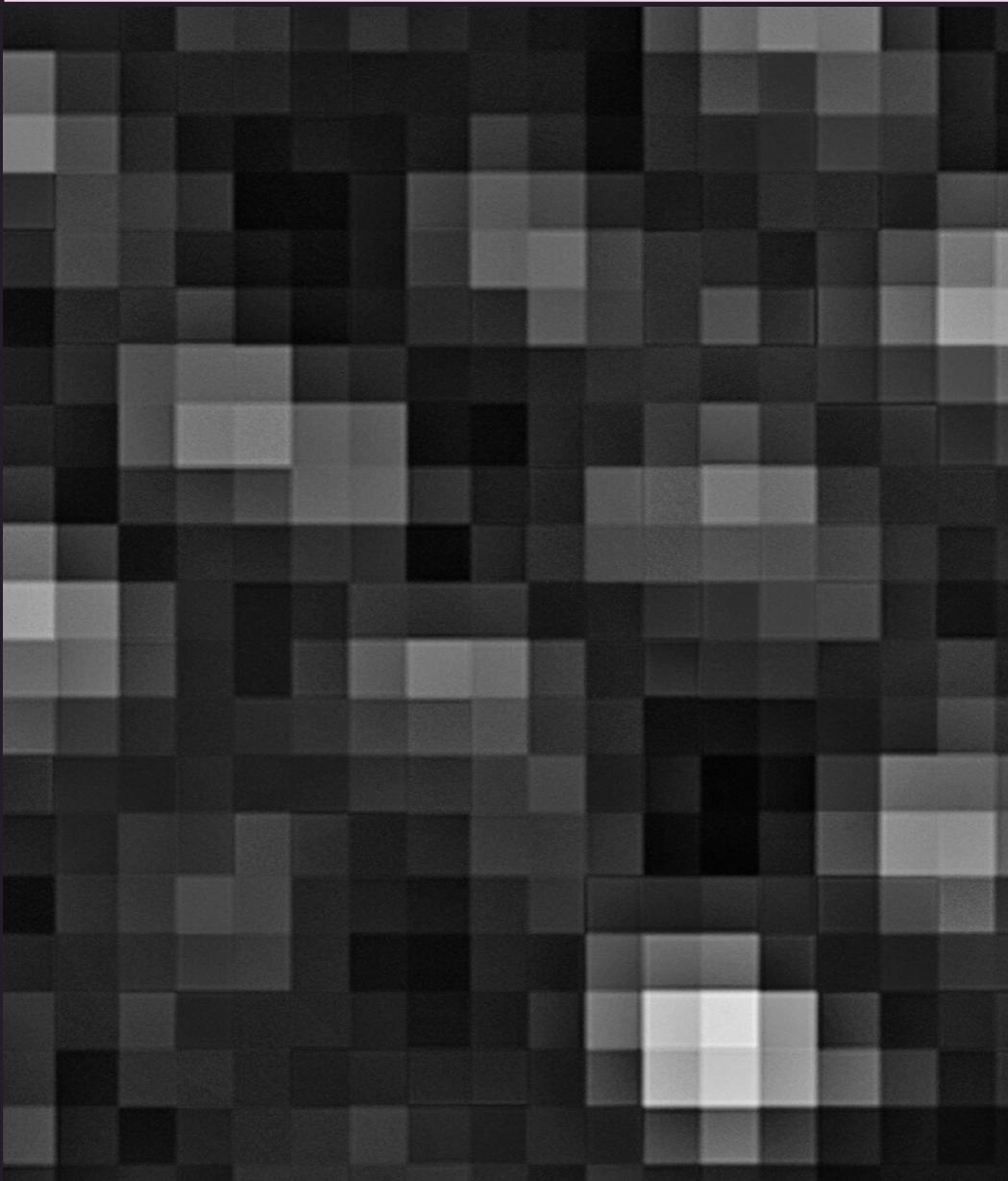
## Absolute

- PX
- PT
- CM
- IN
- MM

# FONT FAMILY



# Absolute Units



**PX - BY FAR THE MOST  
COMMONLY USED ABSOLUTE UNIT**

1px does not necessarily equal the width  
of exactly one pixel!

Not recommended for responsive websites.

**em**



## **EM'S ARE RELATIVE UNITS**

With font-size, 1em equals the font-size of the parent. 2em's is twice the font-size of the parent, etc.

With other properties, 1em is equal to the computed font-size of the element itself.

# rem



## ROOT EMS

Relative to the **root html element's** font-size. Often easier to work with.

If the root font-size is 20px, 1 rem is always 20px, 2rem is always 40px, etc.

# CSS SELECTORS

A SUPER IMPORTANT PART OF CSS!

# CSS RULES

Everything you do in CSS follows this basic pattern:

```
selector {  
    property: value;  
}
```

# CSS RULES

Everything you do in CSS follows this basic pattern:



```
selector {  
  property: value;  
}
```

# CSS RULES

Make all `<h1>` elements purple

```
h1 {  
    color: purple;  
}
```

# FANCIER!

Select every other text input  
and give it a red border:

```
input[type="text"]:nth-of-type(2n){  
    border:2px solid red;  
}
```

email      username      city      zipcode

# UNIVERSAL SELECTOR

*Select everything!*

```
* {  
    color: black;  
}
```

# ELEMENT SELECTOR

*Select all images*

```
img {  
    width: 100px;  
    height: 200px;  
}
```

# SELECTOR LIST

*Select all h1's and h2's*

```
h1, h2 {  
  color: magenta;  
}
```

# CLASS SELECTOR

*Select elements with class of 'complete'*

```
.complete {  
    color: green;  
}
```

# ID SELECTOR

*Select the element with id of 'logout'*

```
#logout {  
    color: orange;  
    height: 200px;  
}
```

# DESCENDANT SELECTOR

*Select all <a>'s that are nested inside an <li>*

```
li a {  
    color: teal;  
}
```

# ADJACENT SELECTOR

*Select only the paragraphs that are immediately preceded by an <h1>*

```
h1 + p {  
    color: red;  
}
```

# DIRECT CHILD

Select only the <li>'s that are direct children of a <div> element

```
div > li {  
  color: white;  
}
```

# ATTRIBUTE SELECTOR

*Select all input elements where the type attribute is set to "text"*

```
input[type="text"] {  
    width: 300px;  
    color: yellow;  
}
```

# PSEUDO CLASSES

keyword added to a selector that specifies a special state of the selected element(s)

- :active
- :checked
- :first
- :first-child
- :hover
- :not()
- :nth-child()
- :nth-of-type()

# PSEUDO ELEMENTS

Keyword added to a selector that lets you style a particular part of selected element(s)

- ::after
- ::before
- ::first-letter
- ::first-line
- ::selection

What happens when  
conflicting styles  
target the same  
elements?

# THE CASCADE

The order your styles are declared in  
and linked to matters!



```
h1 {  
  color: red;  
}  
h1 {  
  color: purple;  
}
```

Purple wins!

# SPECIFICITY

Specificity is how the browser decides which rules to apply when multiple rules could apply to the same element.

It is a measure of how specific a given selector is. The more specific selector "wins"

# SPECIFICITY



```
p {  
  color: yellow;  
}
```

Element Selector



```
section p {  
  color: teal;  
}
```

Element Selector  
+ Element Selector



**ID >**

**CLASS >**

**ELEMENT**



```
section p {  
  color: teal;  
}
```



---

ID Selectors



Class,  
Attribute, &  
Pseudo-Class  
Selectors



---

Element and  
Pseudo-Element  
Selectors

# 1

---

ID Selectors



Class,  
Attribute, &  
Pseudo-Class  
Selectors

```
#submit {  
  color: olive;  
}
```



Element and  
Pseudo-Element  
Selectors



---

ID Selectors



---

Class,  
Attribute, &  
Pseudo-Class  
Selectors



---

Element and  
Pseudo-Element  
Selectors



```
nav a.active {  
    color: orange;  
}
```

# INLINE STYLES

---

Inline Styles

---

ID Selectors

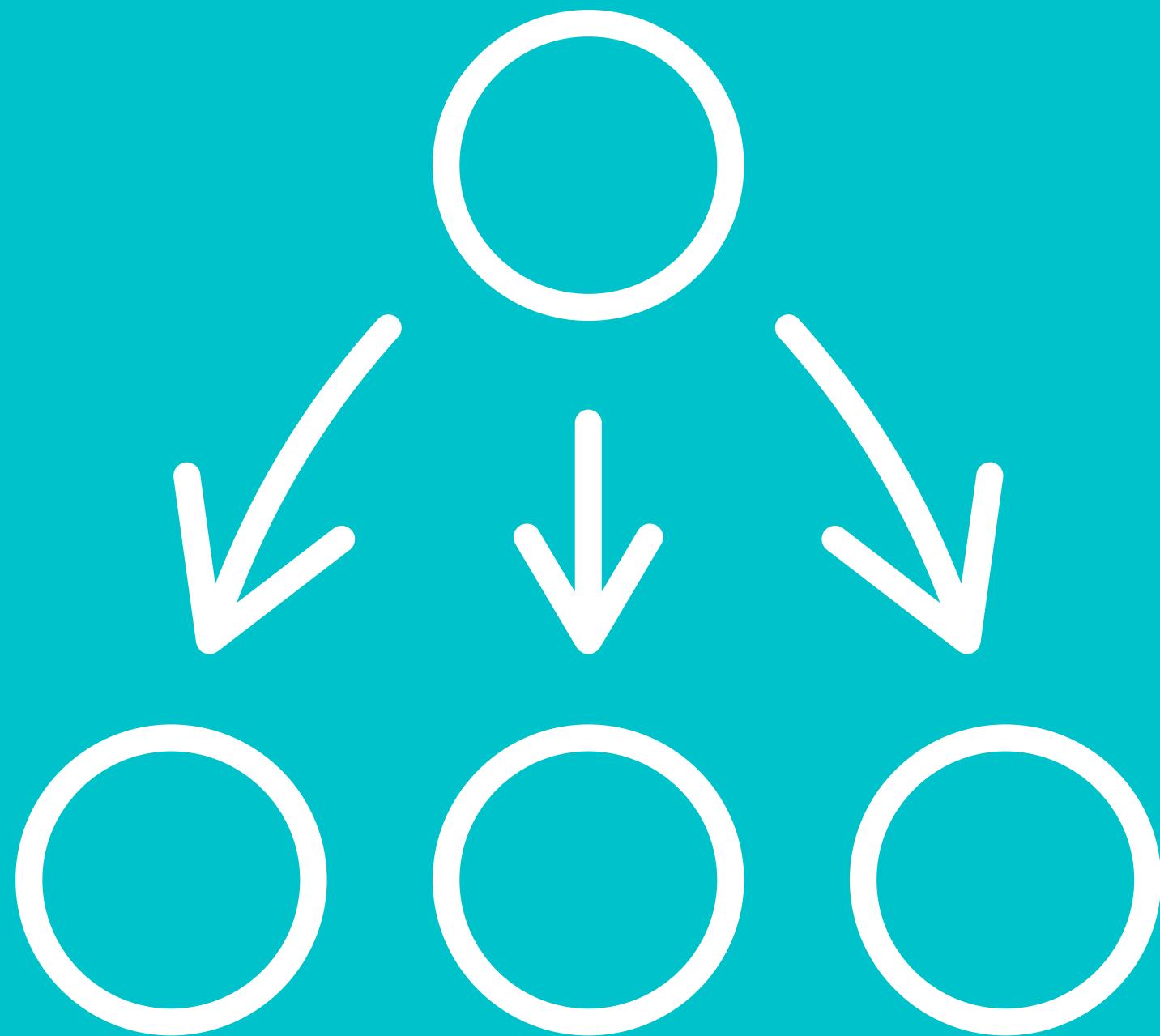
---

Class,  
Attribute, &  
Pseudo-Class  
Selectors

---

Element and  
Pseudo-Element  
Selectors

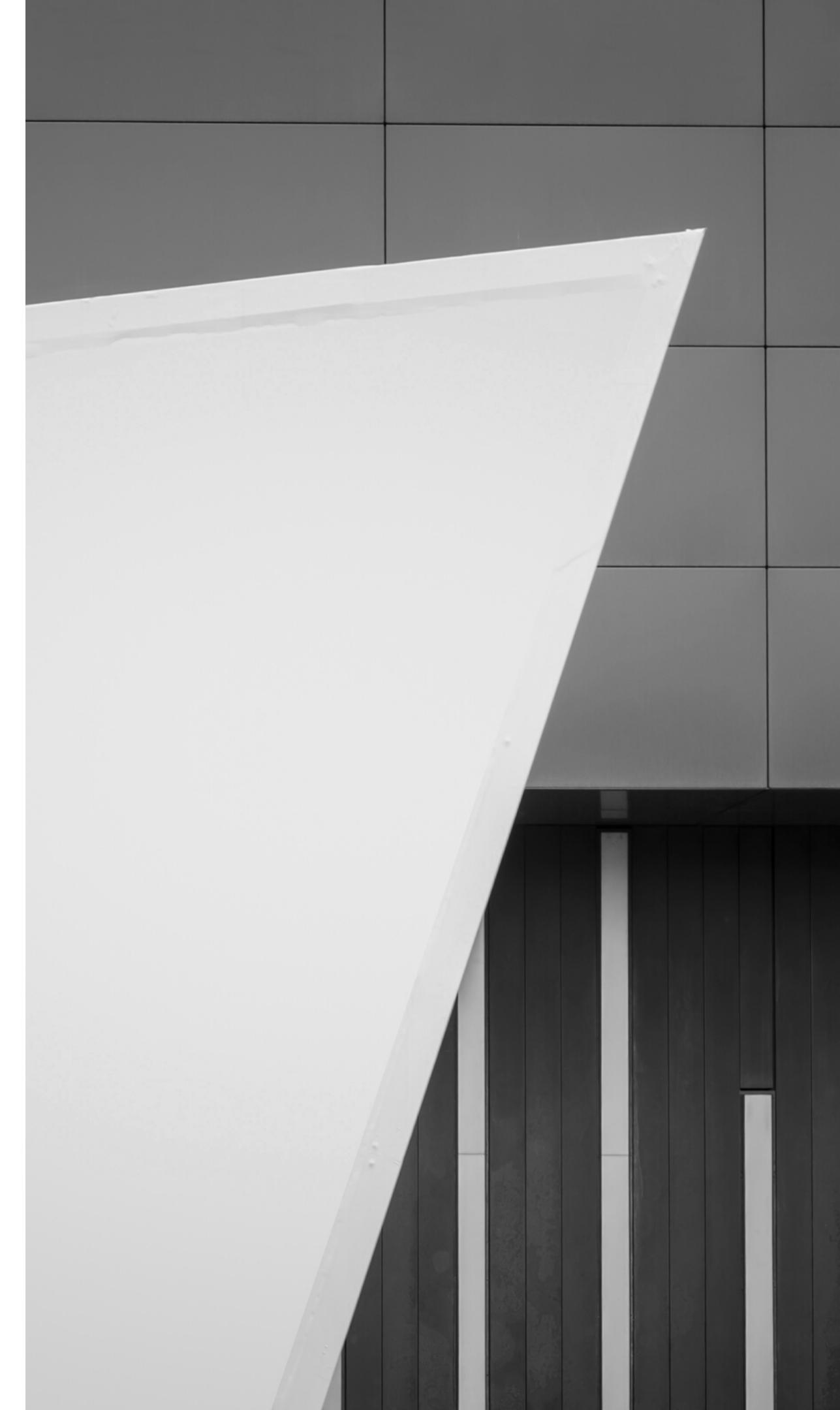
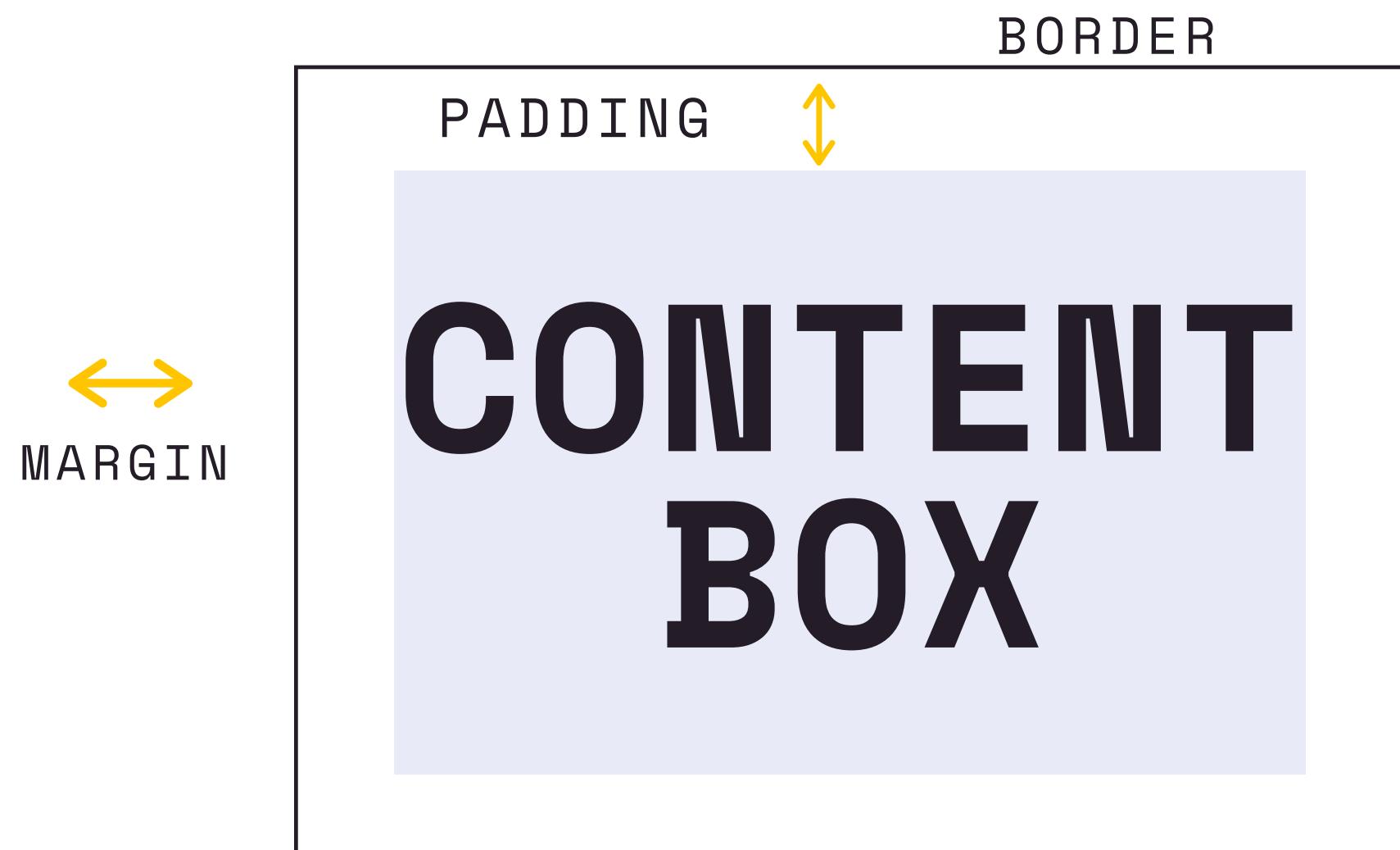
# INHERITANCE



# THE CSS BOX MODEL

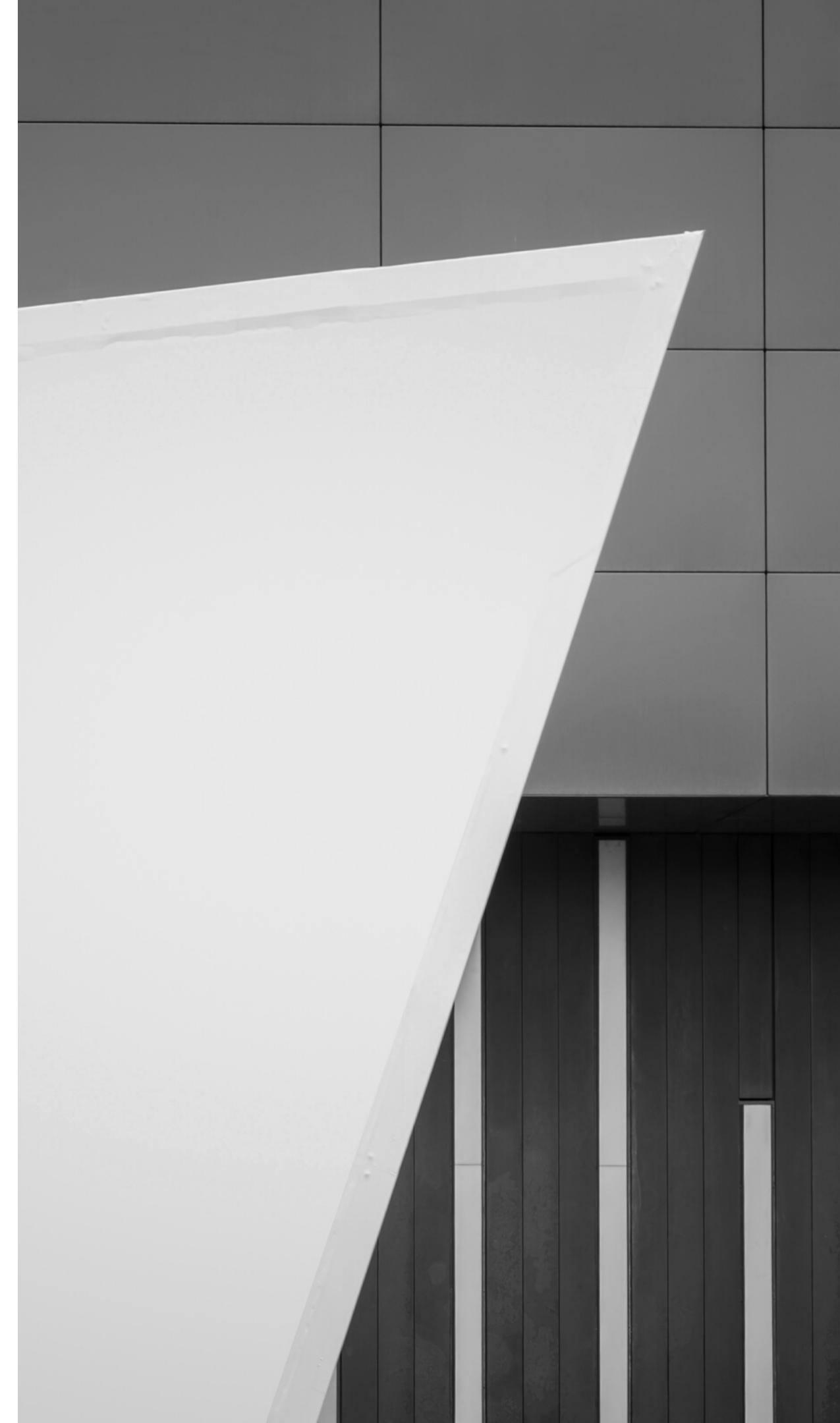
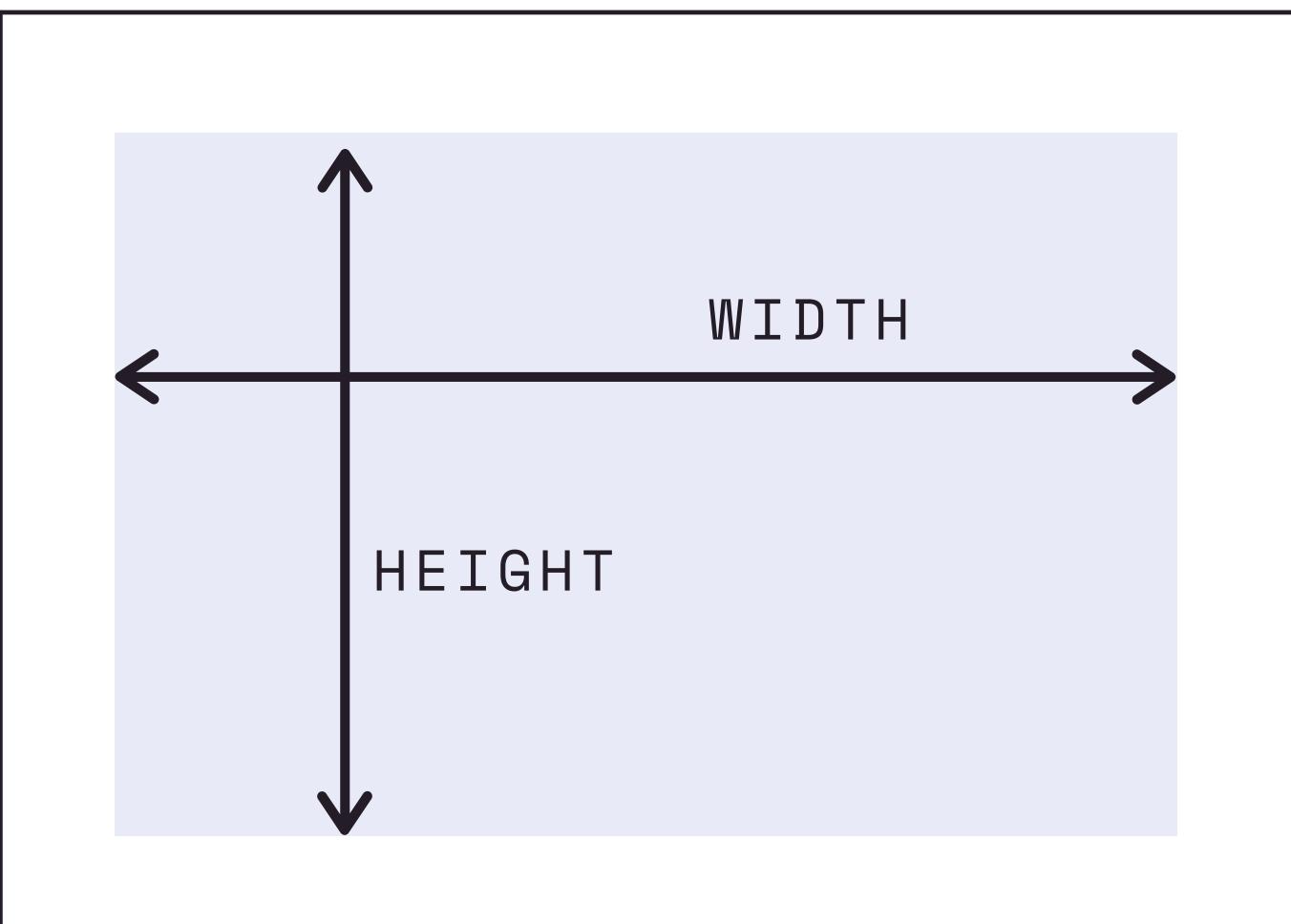


## The Box Model



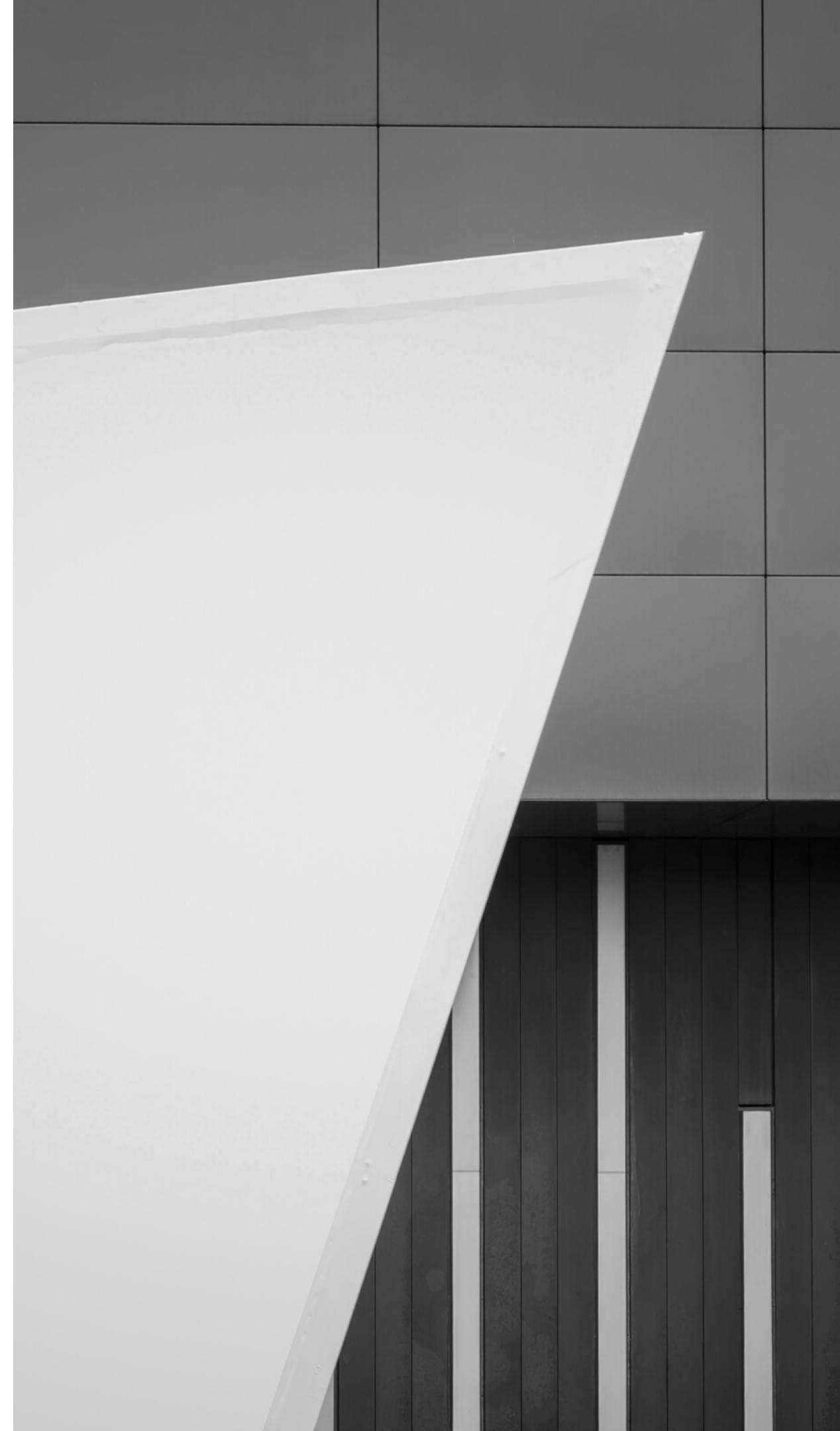
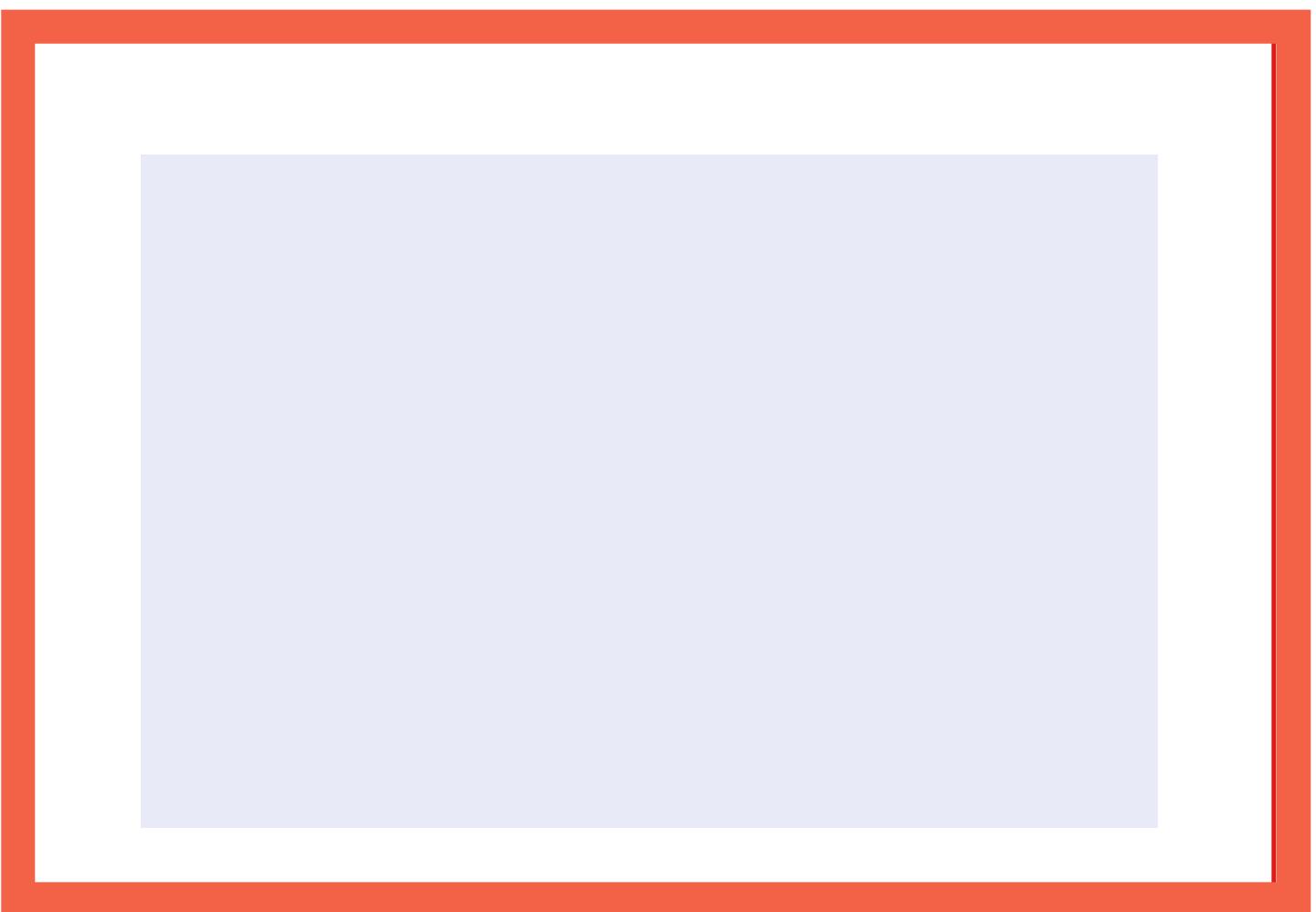


## Width & Height





## Border



# Border Properties

(the important ones)

## BORDER-WIDTH

Controls the thickness of the border.

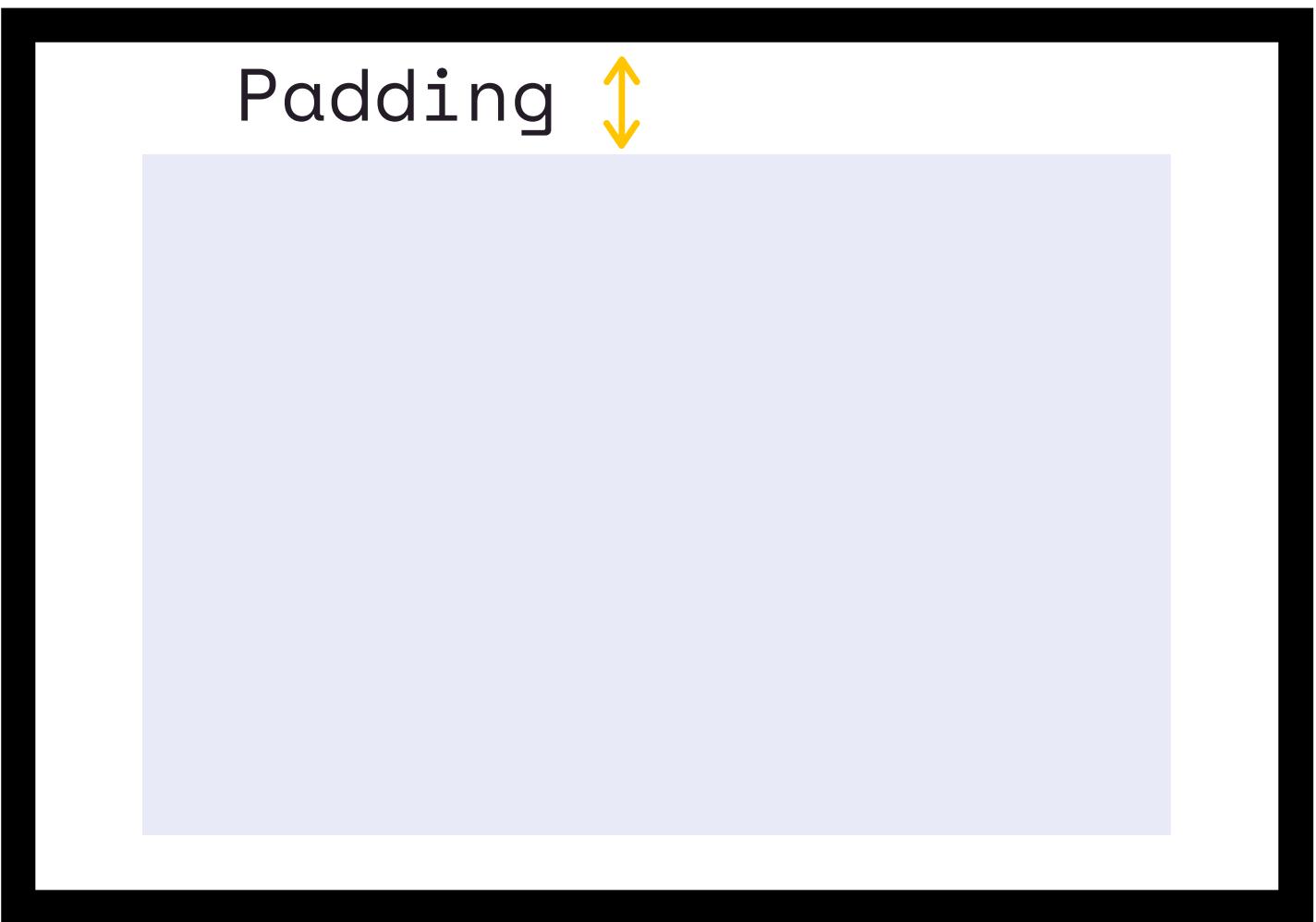
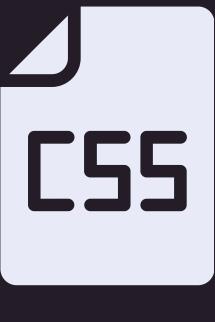
## BORDER-COLOR

Controls the...color of the border

## BORDER-STYLE

Controls the line style - dashed, solid, etc.





```
="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-sca  
ef="/favicon.ico" type="image/x-icon">  
icon.ico" type="image/x-icon">  
  
'stylesheet' href="css/materialize.min.css" media="screen, print">  
f="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/  
ef="/css/animate.css">  
ef="/css/theme.css">  
  
rapper">  
tainer">  
class="brand-logo hide-on-med-and-up"><span style="font-size:  
  
ainer">  
ow">  
12 m8 l8 hide-on-small-only">  
ight:400">starter</span><span style="font-size:  
stup network and workspace tab1">  
span id="banner-new">
```

# Individual Properties

PADDING - LEFT

PADDING - RIGHT

PADDING - BOTTOM

PADDING - TOP

# Shorthand Property

SET ALL FOUR SIDES AT ONCE !



# Padding Shorthand Property

Apply to all four sides

`padding: 10px;`

vertical | horizontal

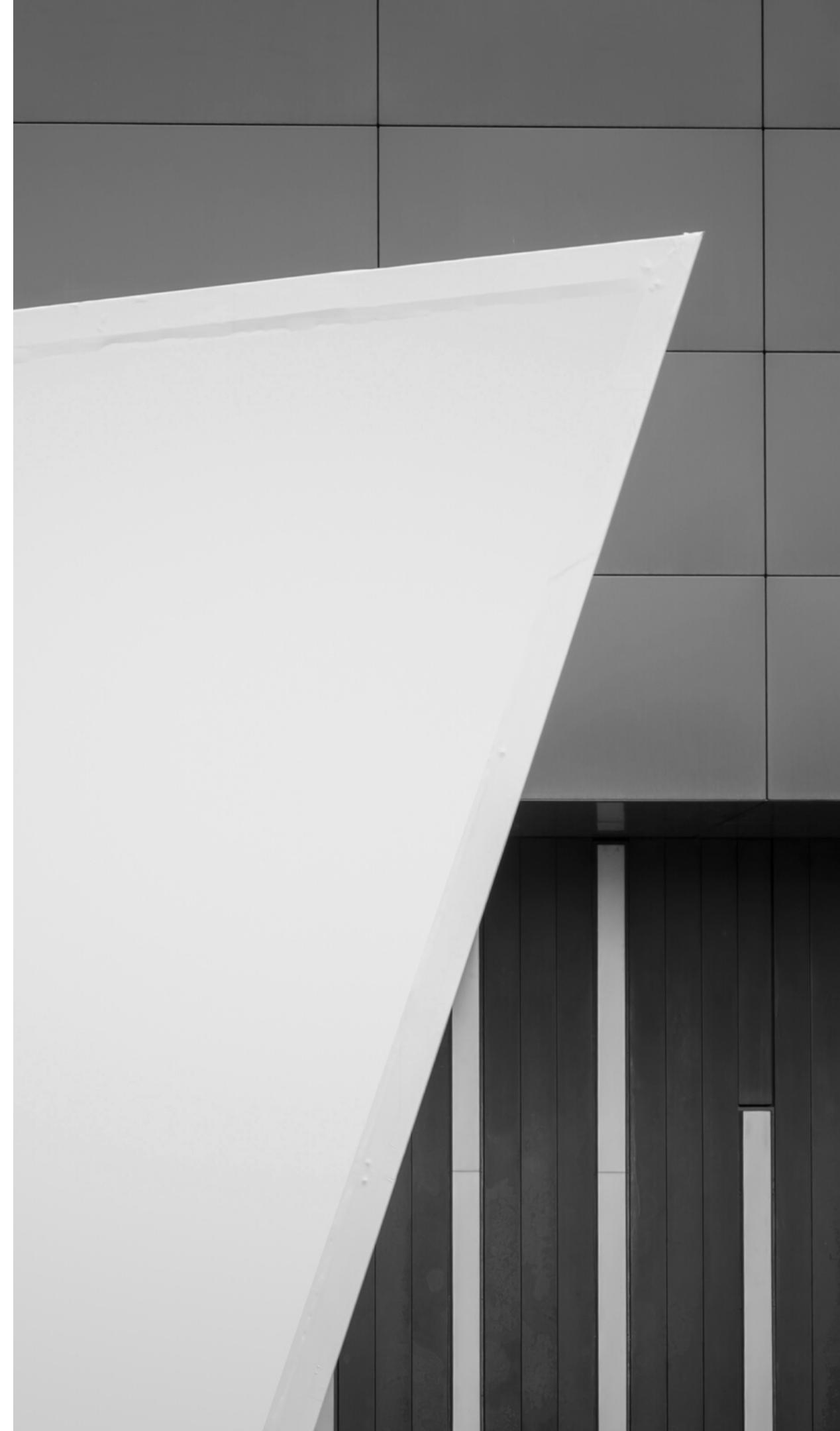
`padding: 5px 10px;`

top | horizontal | bottom

`padding: 1px 2px 2px;`

top | right | bottom | left

`padding: 5px 1px 0 2px;`



```
="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-sca  
ef="/favicon.ico" type="image/x-icon">  
icon.ico" type="image/x-icon">  
  
'stylesheet' href="css/materialize.min.css" media="screen, print">  
f="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/  
ef="/css/animate.css">  
ef="/css/theme.css">  
  
rapper">  
tainer">  
class="brand-logo hide-on-med-and-up"><span style="font-size:  
  
ainer">  
ow">  
12 m8 l8 hide-on-small-only">  
ight:400">starter</span><span style="font-size:  
up network and workspace tab!>  
span id="banner-create">  
nnect to your workspace </span>
```

# Individual Properties

MARGIN-LEFT

MARGIN-RIGHT

MARGIN-BOTTOM

MARGIN-TOP

# Shorthand Property

SET ALL FOUR SIDES AT ONCE !

# Margin Shorthand Property

Apply to all four sides

`margin: 10px;`

vertical | horizontal

`margin: 5px 10px;`

top | horizontal | bottom

`margin: 1px 2px 2px;`

top | right | bottom | left

`margin: 5px 1px 0 2px;`

# Display Property



# INLINE ELEMENTS



- Inline elements fit in alongside other elements
- Block level elements take up a whole "block" of space.

# BLOCK ELEMENTS



# Display Property

(our first encounter)

## INLINE

Width & Height are ignored. Margin & padding push elements away horizontally but not vertically.

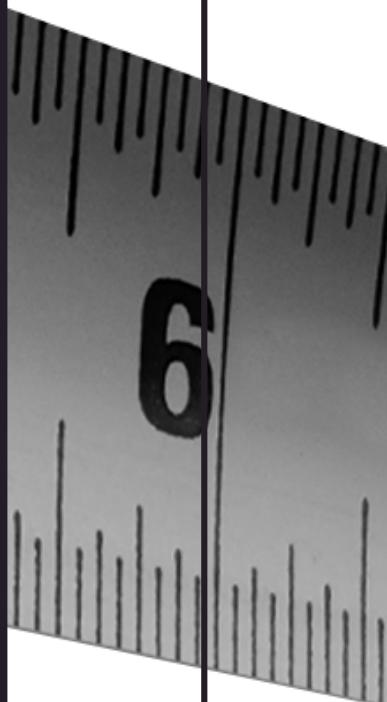
## BLOCK

Block elements break the flow of a document. Width, Height, Margin, & Padding are respected.

## INLINE-BLOCK

Behaved like an inline element except Width, Height, Margin, & Padding are respected

WEB DEVELOPER BOOTCAMP



# CSS UNITS



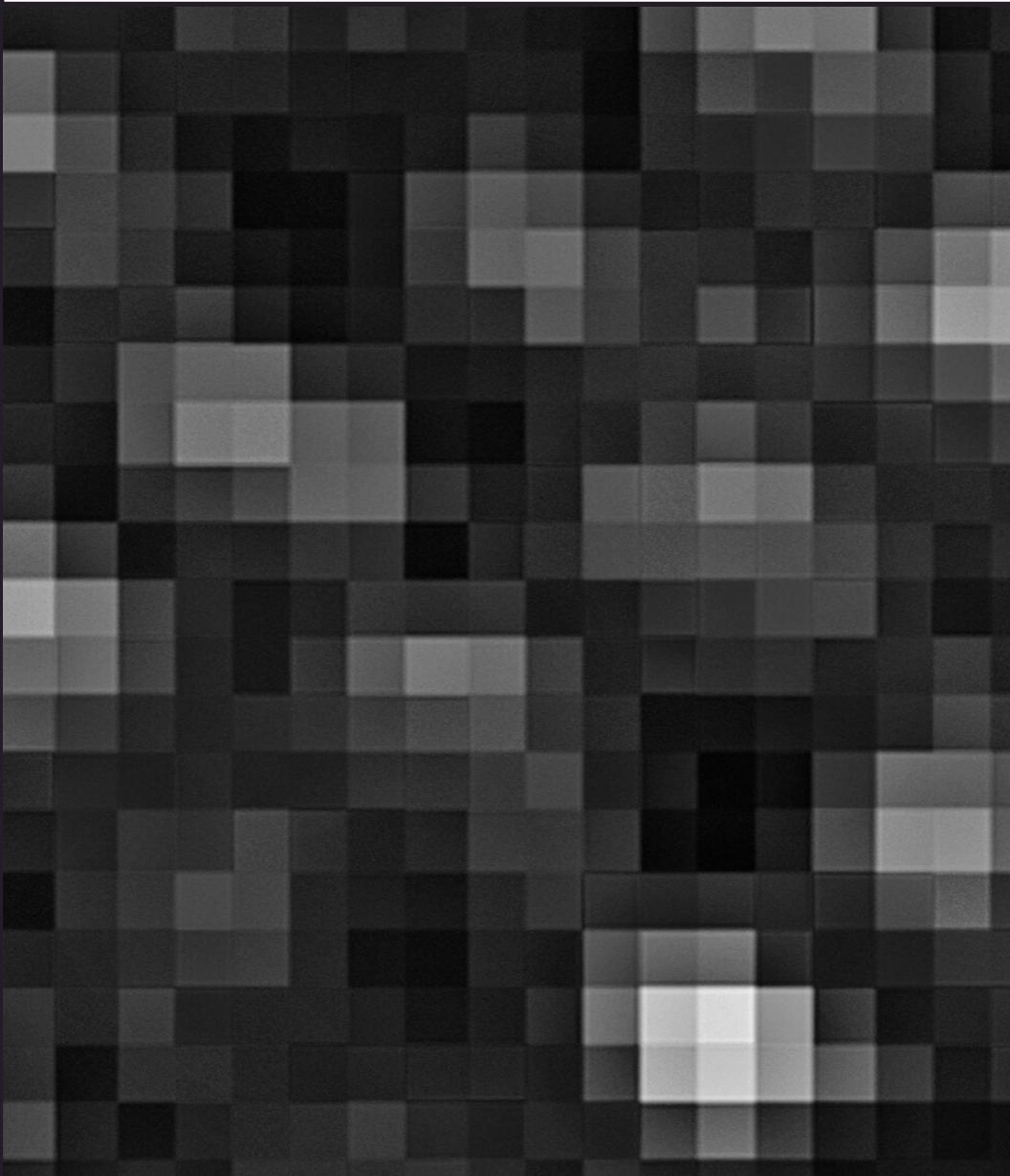
## Relative

- EM
- REM
- VH
- VW
- %
- AND MORE!

## Absolute

- PX
- PT
- CM
- IN
- MM

# Absolute Units



**PX - BY FAR THE MOST  
COMMONLY USED ABSOLUTE UNIT**

1px does not necessarily equal the width  
of exactly one pixel!

Not recommended for responsive websites.

# percentages



**PERCENTAGES ARE ALWAYS  
RELATIVE TO SOME OTHER VALUE**

Sometimes, it's a value from the parent  
and other times it's a value from the  
element itself.

**width: 50%** - half the width of the parent

**line-height: 50%** - half the font-size of  
the element itself

**em**



## **EM'S ARE RELATIVE UNITS**

With font-size, 1em equals the font-size of the parent. 2em's is twice the font-size of the parent, etc.

With other properties, 1em is equal to the computed font-size of the element itself.

# rem

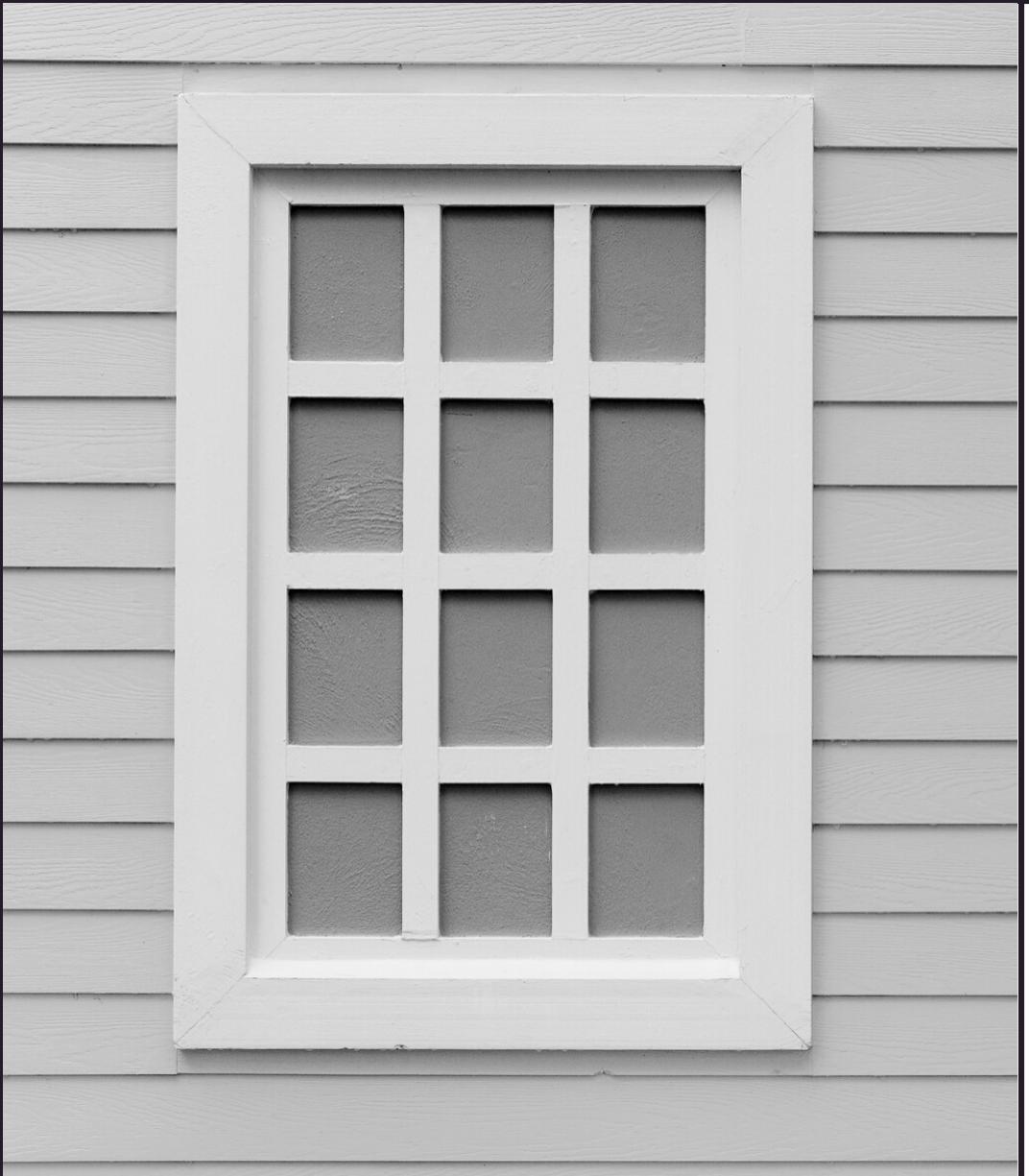


## ROOT EMS

Relative to the **root html element's** font-size. Often easier to work with.

If the root font-size is 20px, 1 rem is always 20px, 2rem is always 40px, etc.

# **vw & vh**



## **VIEW HEIGHT & VIEW WIDTH**

**1vw** is 1% of the width of the viewport

**1vh** is 1% of the height of the viewport

For example: `height: 100vh` would make an element take up the full height on screen



**Opacity +**  
**Alpha Channel**

# POSITION



**rgba(0, 209, 112, 0.5)**

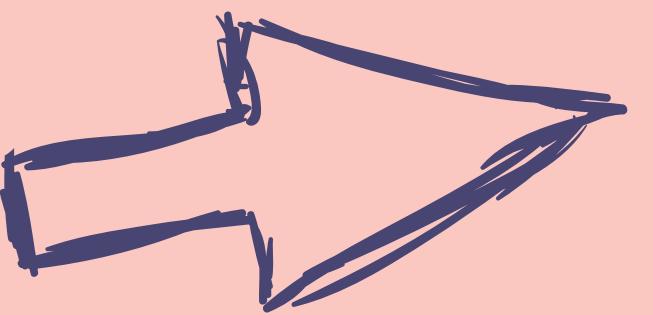
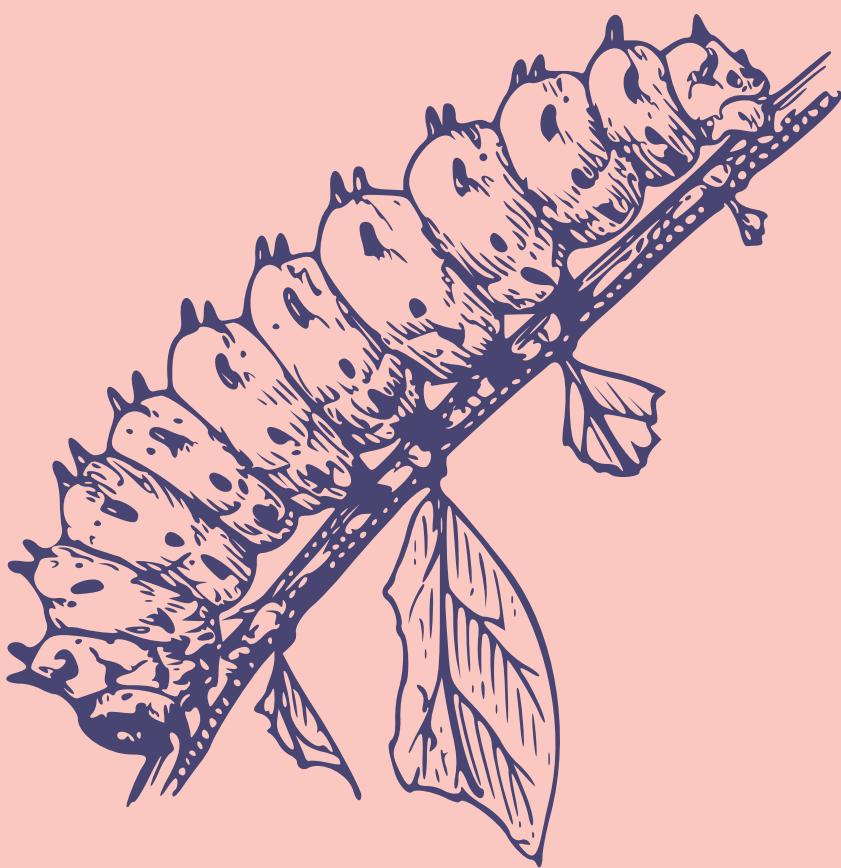
red

green

blue

alpha

# Transitions



# Transition:

PROPERTY NAME | DURATION | TIMING FUNCTION | DELAY

# Transform





```
/* Function values */
transform: matrix(1.0, 2.0, 3.0, 4.0, 5.0, 6.0);
transform: matrix3d(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);
transform: perspective(17px);
transform: rotate(0.5turn);
transform: rotate3d(1, 2.0, 3.0, 10deg);
transform: rotateX(10deg);
transform: rotateY(10deg);
transform: rotateZ(10deg);
transform: translate(12px, 50%);
transform: translate3d(12px, 50%, 3em);
transform: translateX(2em);
transform: translateY(3in);
transform: translateZ(2px);
transform: scale(2, 0.5);
transform: scale3d(2.5, 1.2, 0.3);
transform: scaleX(2);
transform: scaleY(0.5);
transform: scaleZ(0.3);
transform: skew(30deg, 20deg);
transform: skewX(30deg);
transform: skewY(1.07rad);

/* Multiple function values */
transform: translateX(10px) rotate(10deg) translateY(5px);
```



background

GOOGLE FONTS

# INTRODUCTION TO CSS FLEXBOX

You'll love it!

# The Basics

## WHAT IS IT?

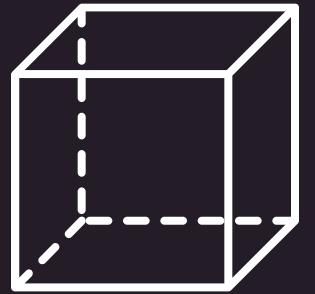
Flexbox is a one-dimensional layout method for laying out items in rows or columns

## IT'S NEW(ISH)

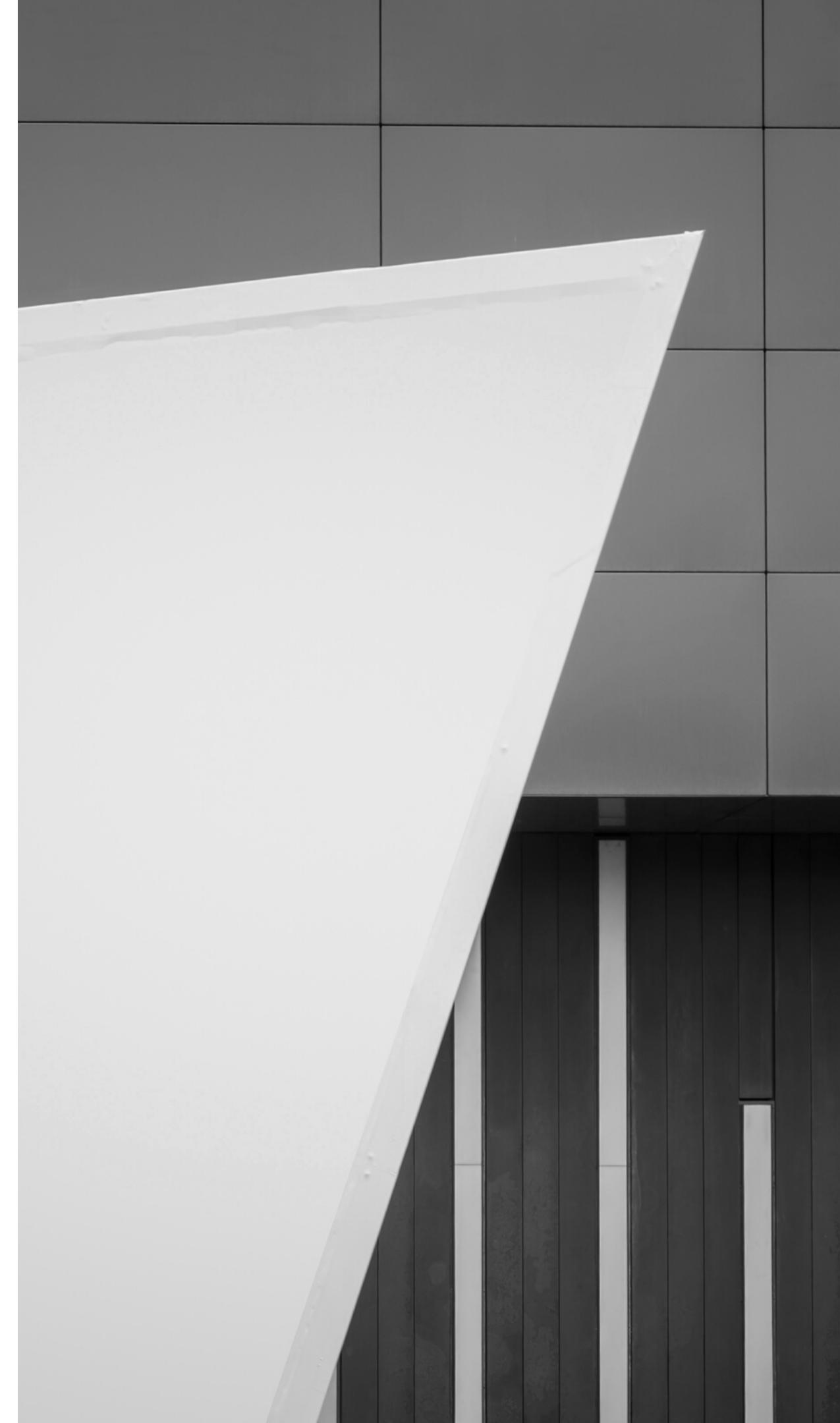
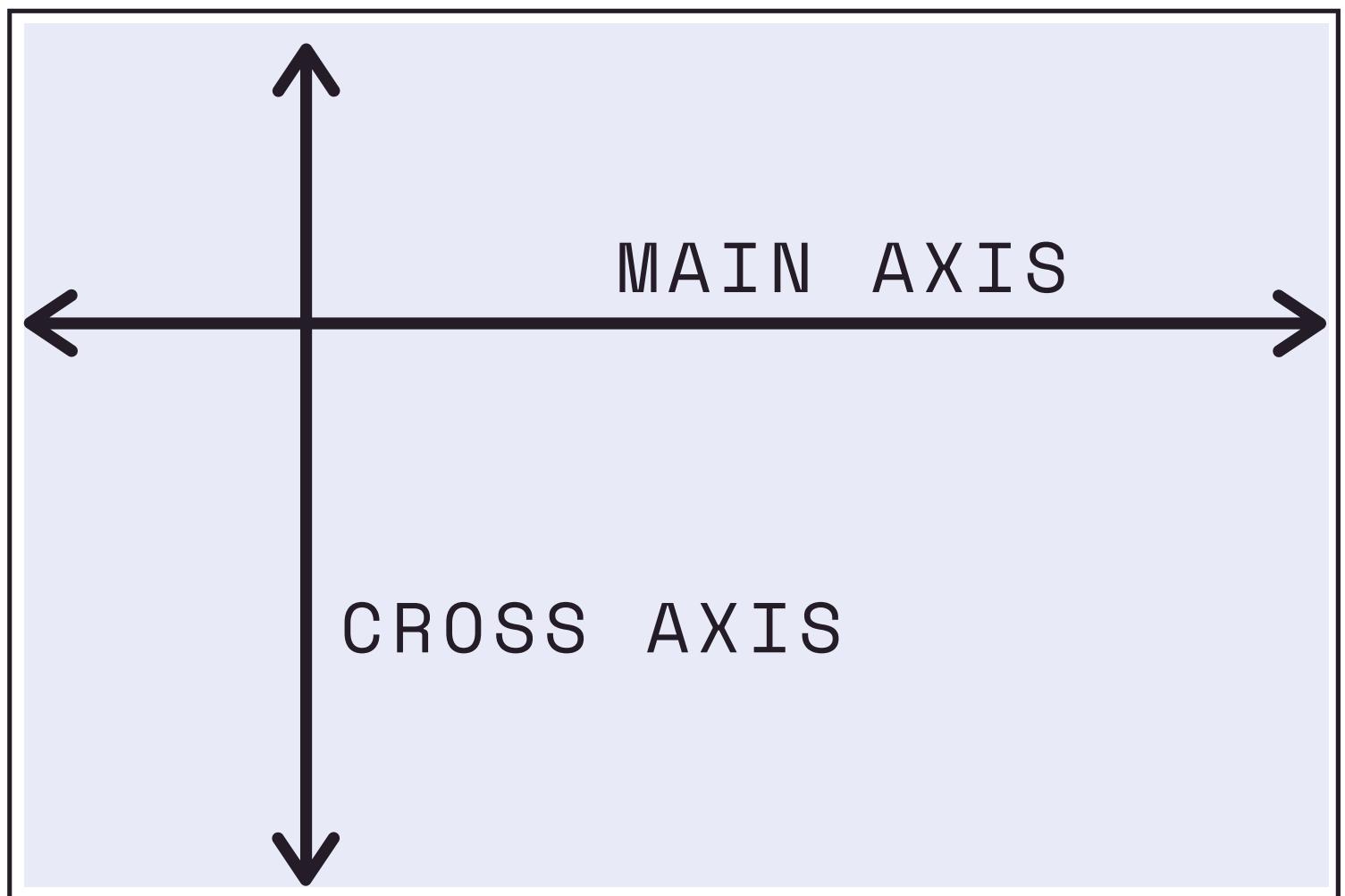
Flexbox is a recent addition to CSS, included to address common layout frustrations

## WHY 'FLEX'?

Flexbox allows us to distribute space dynamically across elements of an unknown size, hence the term "flex"



## The Flex Model



# FLEX DIRECTION



`flex-direction: row;`

# FLEX DIRECTION

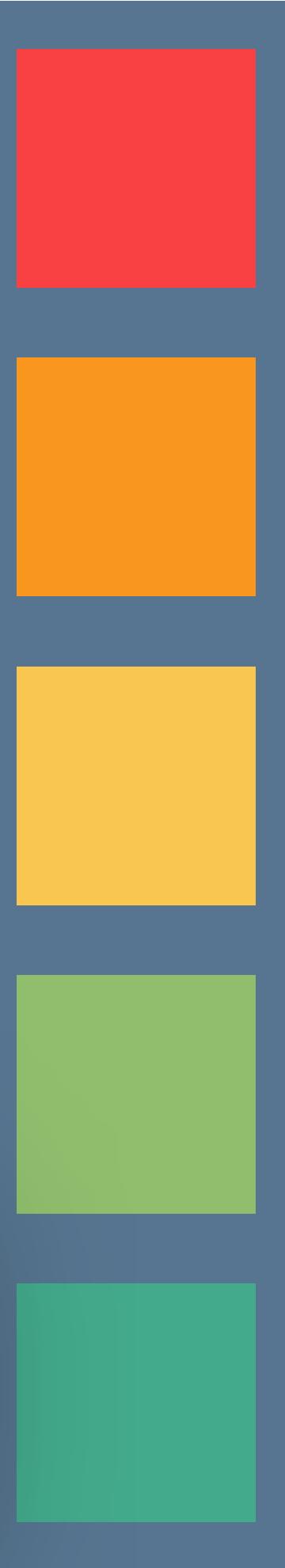
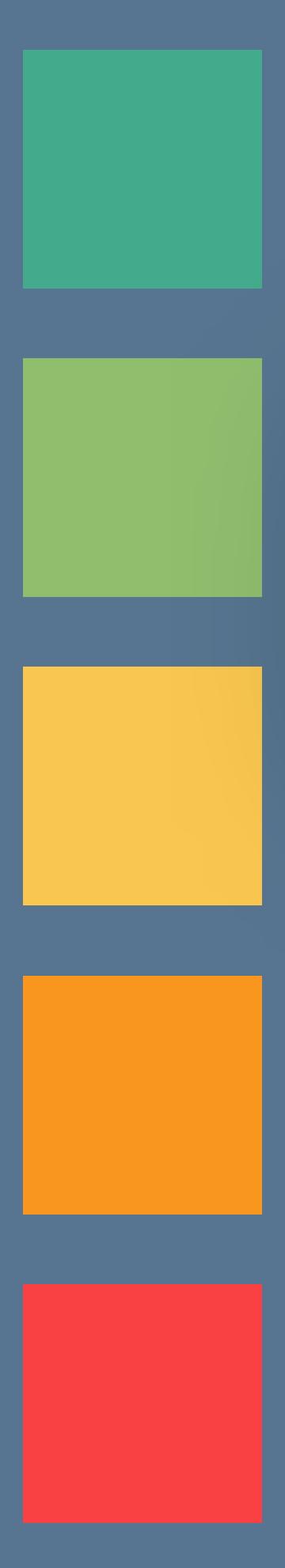


`flex-direction: row-reverse;`

# FLEX DIRECTION



```
flex-direction: column;
```



```
flex-direction: column-reverse;
```

# FLEX WRAP



```
flex-wrap: wrap;
```

# JUSTIFY CONTENT



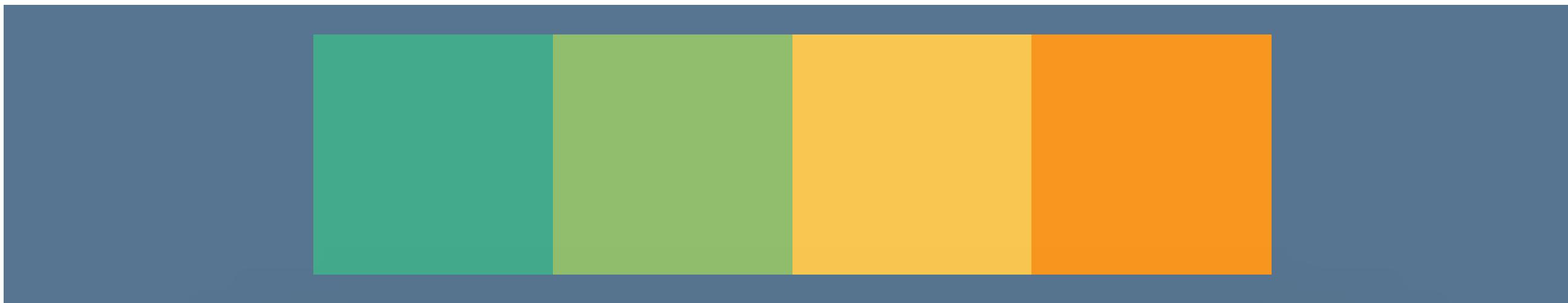
```
justify-content: flex-start;
```

# JUSTIFY CONTENT



```
justify-content: flex-end;
```

# JUSTIFY CONTENT



```
justify-content: center;
```

# JUSTIFY CONTENT



`justify-content: space-between;`

# JUSTIFY CONTENT



`justify-content: space-around;`

# ALIGN ITEMS



```
align-items: flex-start;
```

# ALIGN ITEMS



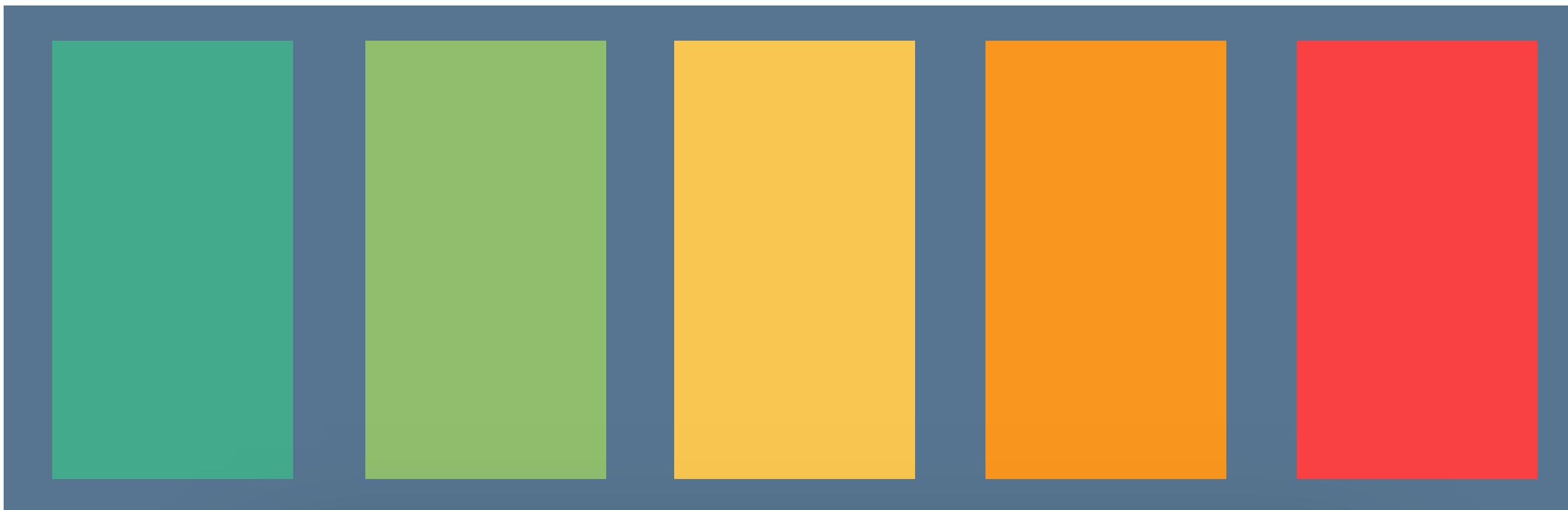
```
align-items: flex-end;
```

# ALIGN ITEMS



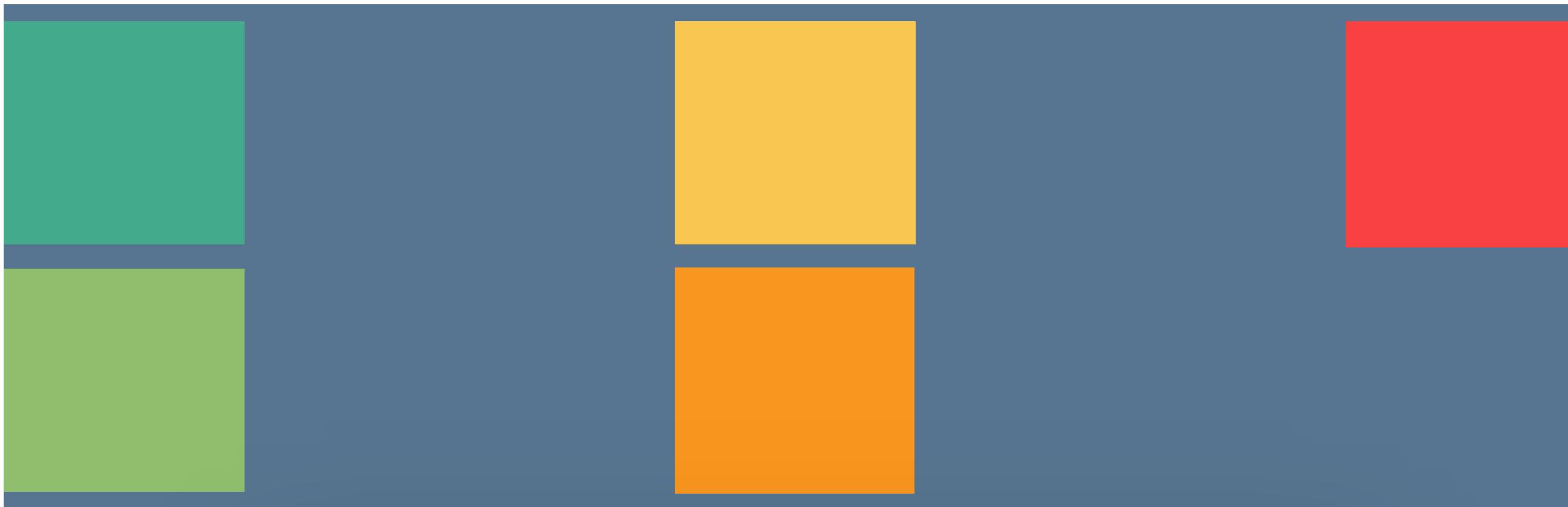
```
align-items: center;
```

# ALIGN ITEMS



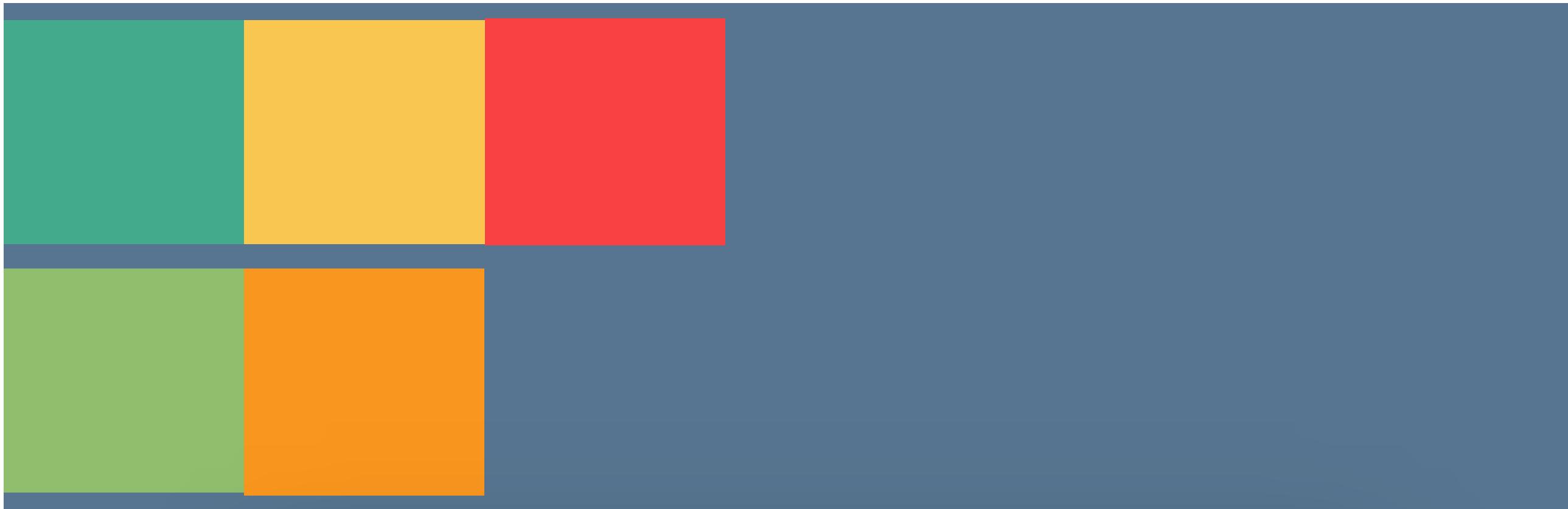
```
align-items: stretch;
```

# ALIGN CONTENT



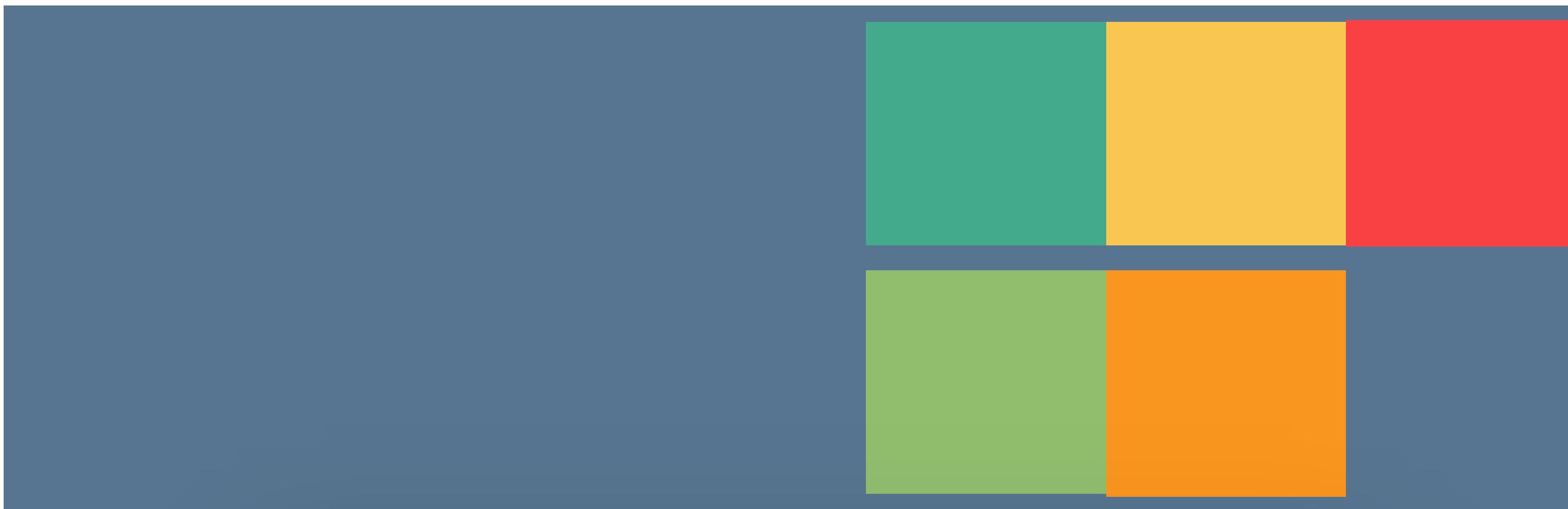
align-content:space-between;

# ALIGN CONTENT



```
align-content: flex-start;
```

# ALIGN CONTENT



```
align-content: flex-end;
```

# ALIGN CONTENT



```
align-content:center;
```

# ALIGN SELF



align-self: flex-end;

# Flex Sizing Properties

## **FLEX-BASIS**

Defines the initial size of an element before additional space is distributed.

## **FLEX-GROW**

Controls the amount of available space an element should take up.  
Accepts a unit-less number value.

## **FLEX-SHRINK**

If items are larger than the container, they shrink according to flex-shrink.

# RESPONSIVE DESIGN

what is it & why you should care

## THE PROBLEM

As mobile devices and tablets became widely available, developers had a problem...how do we create websites that look good on all screen sizes?

## ONE APPROACH

Early on, it was common to create separate stylesheets for different devices, or even completely different websites for each size.

## ENTER RESPONSIVE

These days, we typically create ONE website and stylesheet that is able to respond to different device sizes and features.

# MEDIA QUERIES

Media queries allow us to modify our styles depending on particular parameters like screen width or device type.





```
@media (max-width: 800px) {  
  .sidebar {  
    display: none;  
  }  
  
  .main {  
    width: 80%;  
  }  
}  
  
@media (min-width: 30em) and (orientation: landscape) {  
  #container {  
    flex-direction: column;  
    justify-content: center;  
  }  
}
```

# MEDIA QUERIES



# Bootstrap

---

"THE WORLD'S MOST POPULAR CSS FRAMEWORK"

Web Developer Bootcamp

# JavaScript Basics

VALUES & VARIABLES



FRONT END

BACK END

# Primitive Types

## THE BASIC BUILDING BLOCKS

Number

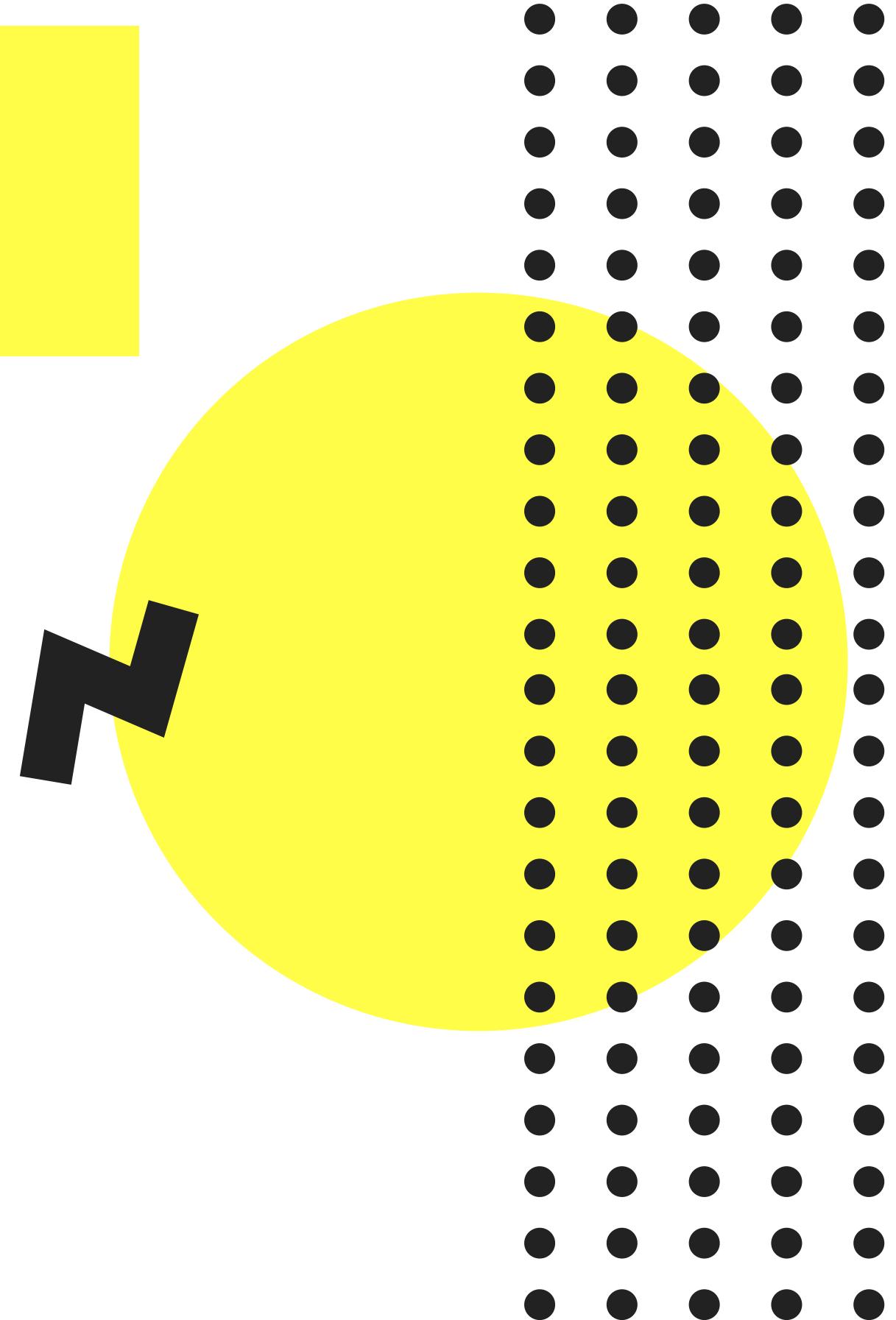
String

Boolean

Null

Undefined

\* Technically there are two others: Symbol and BigInt



# DIFFERENT DATA TYPES

Hall & Oates - When The Morning Comes

426,334 views • Apr 2, 2011

1.7K

88

SHARE

SAVE

...

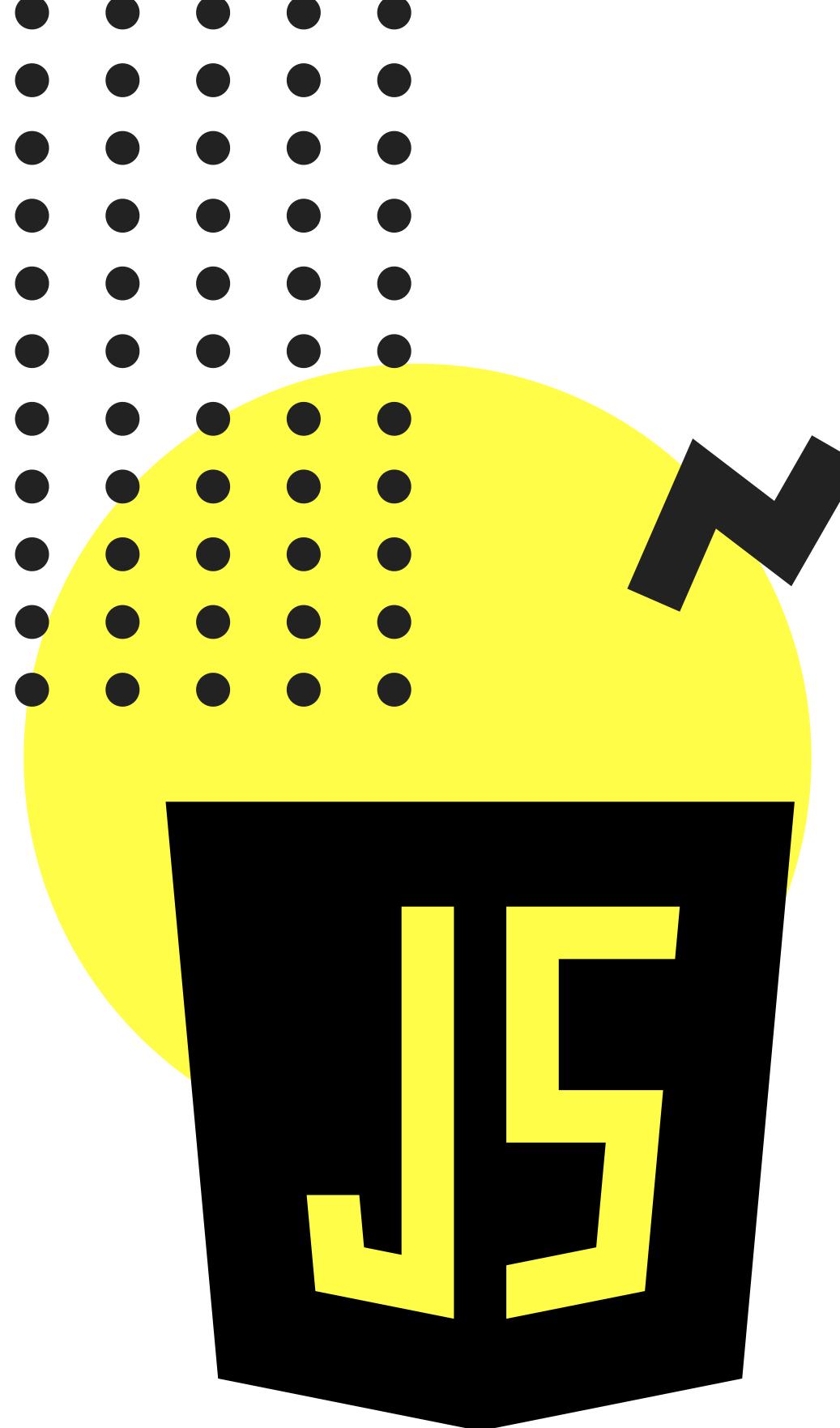


**mickey castle**  
1.61K subscribers

SUBSCRIBE

Best Songs From 1970's Hall & Oates

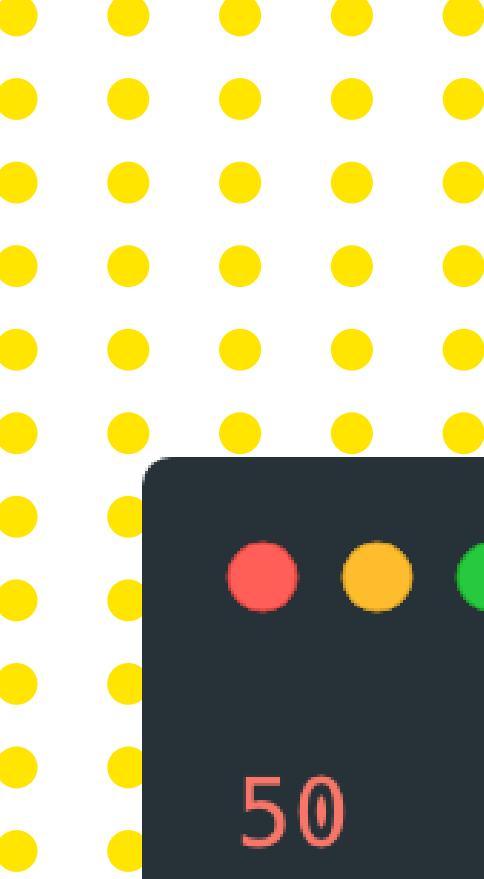
SHOW MORE



# Running Code in The Console

## **THE EASIEST PLACE TO START**

Early on, we'll run our code using the Chrome developer tools console. Then, we'll learn how to write external scripts.



50

7

3.874

0.99

-45

-777.23444

# Numbers

## IN JAVASCRIPT

- JS has one number type
- Positive numbers
- Negatives numbers
- Whole numbers (integers)
- Decimal numbers

# Math Operations



```
//Addition  
50 + 5 //55  
  
//Subtraction  
90 - 1 //89  
  
//Multiplication  
11111 * 7 //77777  
  
//Division  
400 / 25 //16  
  
//Modulo!!  
27 % 2 //1
```

// creates a comment  
(the line is ignored)

# NOT A NUMBER

# Nan

NaN is a numeric value that represents something that is...not a number

N

N

# Not A Number

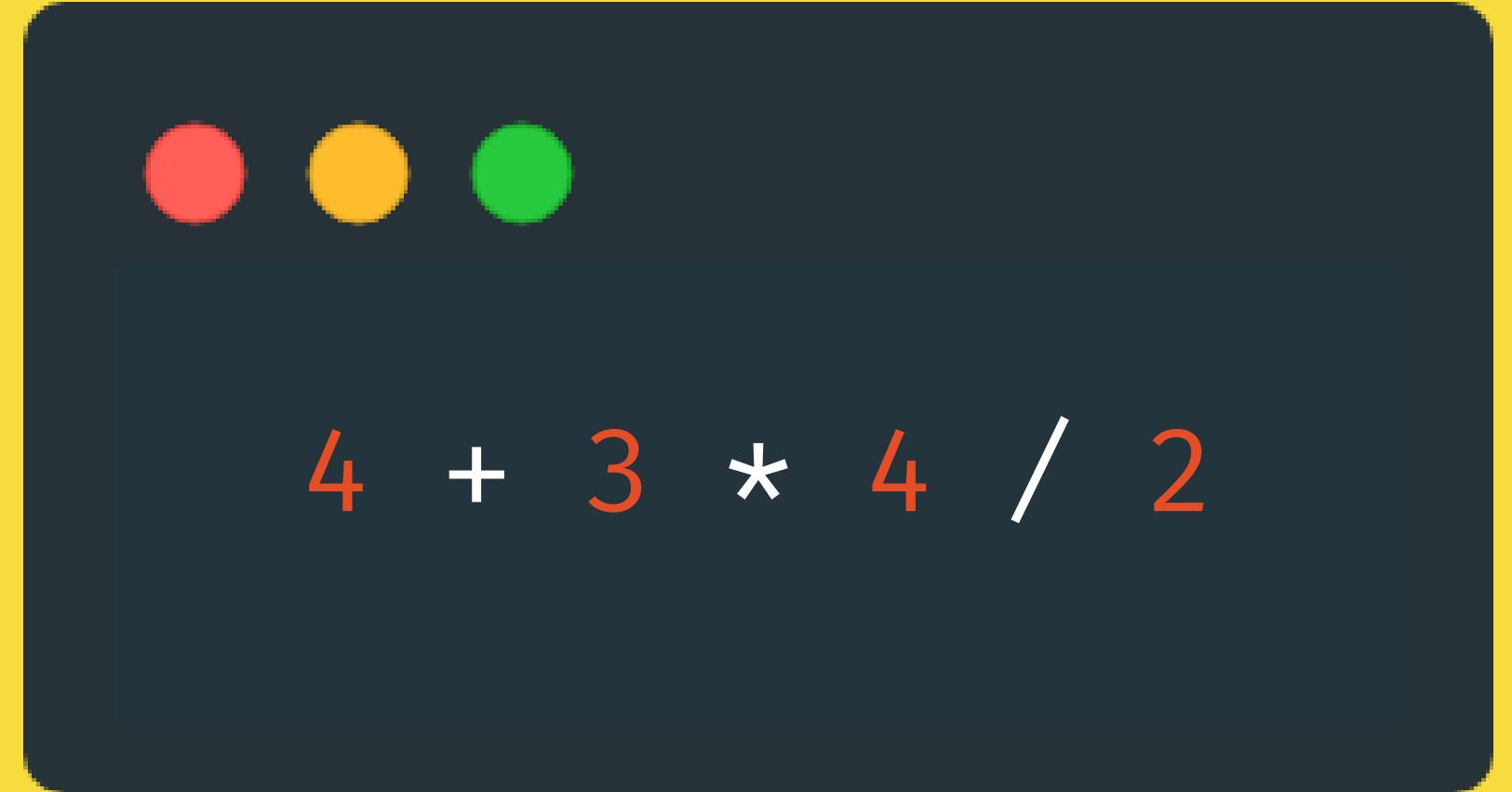


0/0 //NaN

1 + NaN //NaN

• •

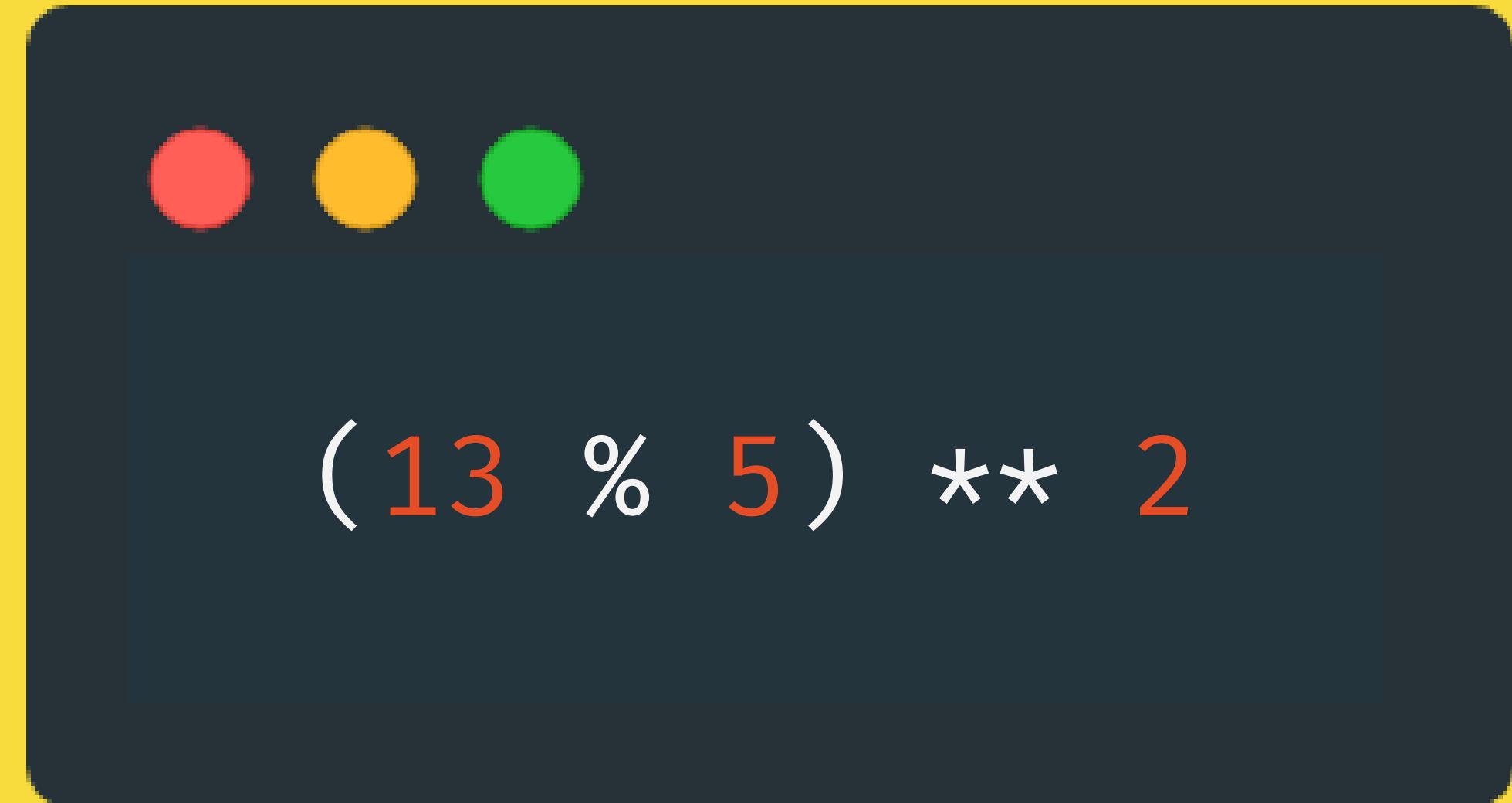
# WHAT DOES THIS EVALUATE TO??



4 + 3 \* 4 / 2

• •

# WHAT DOES THIS EVALUATE TO??



• •

# WHAT DOES THIS EVALUATE TO??



200 + 0/0



# Variables

**VARIABLES ARE LIKE  
LABELS FOR VALUES**

- We can store a value and give it a name so that we can:
  - Refer back to it later
  - Use that value to do...stuff
  - Or change it later one

# BASIC SYNTAX

```
...  
let someName = value;
```

# BASIC SYNTAX



```
let year = 1985;
```

Make me a variable called "year" and give it the value of 1985

# N

# RECALL VALUES



```
let hens = 4;  
  
let roosters = 2;  
  
hens + roosters //6
```

N

# RECALL VALUES



```
let hens = 4;  
//A raccoon killed a hen :(  
hens - 1; //3  
  
hens; //Still 4!
```

This does not change the value stored in hens

```
//To actually change hens:  
hens = hens - 1;  
hens //3
```

This does!

N

# CONST



```
const hens = 4;  
hens = 20; //ERROR!
```

`const` works just like  
`let`, except you CANNOT  
change the value

```
const age = 17;  
age = age + 1; //ERROR!
```

NOT ALLOWED!  
YOU'RE IN TROUBLE!!  
I'M TELLING MOM!!!

# WHY USE CONST?



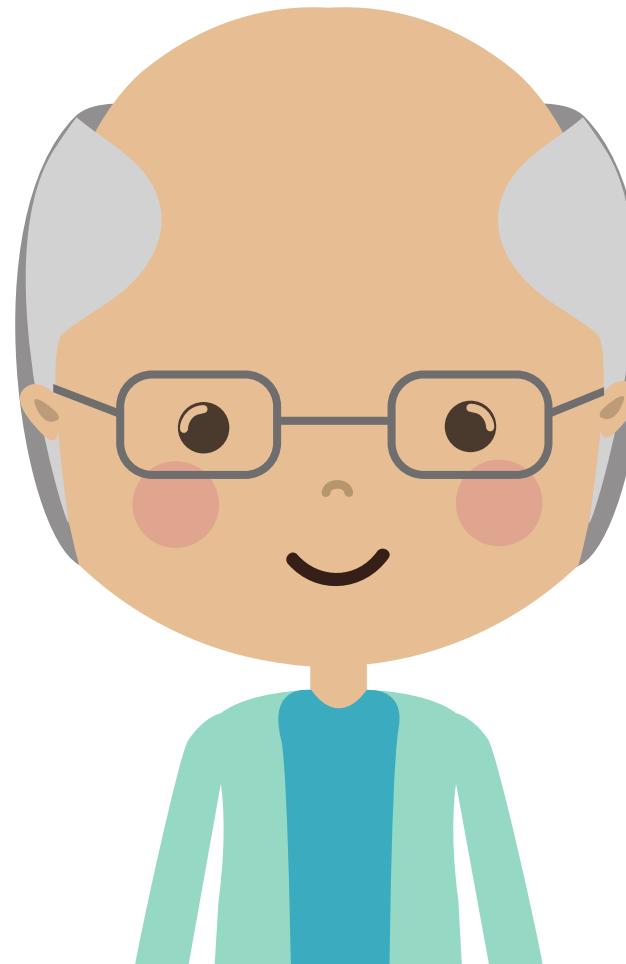
```
const pi = 3.14159;  
  
const daysInWeek = 7;  
  
const minHeightForRide = 60;
```

Once we cover Arrays & Objects, we'll see other situations where *const* makes sense over *let*.

# VAR

## THE OLD VARIABLE KEYWORD

BEFORE LET & CONST, VAR WAS THE ONLY WAY OF DECLARING VARIABLES. THESE DAYS, THERE ISN'T REALLY A REASON TO USE IT.



# What is the value of totalScore?



```
let totalScore = 199;  
totalScore + 1;
```

# What is the value of totalScore?



```
let totalScore = 199;  
totalScore + 1;
```

# What is the value of temperature?



```
const temperature = 83;  
temperature = 85;
```

# What is the value of bankBalance?



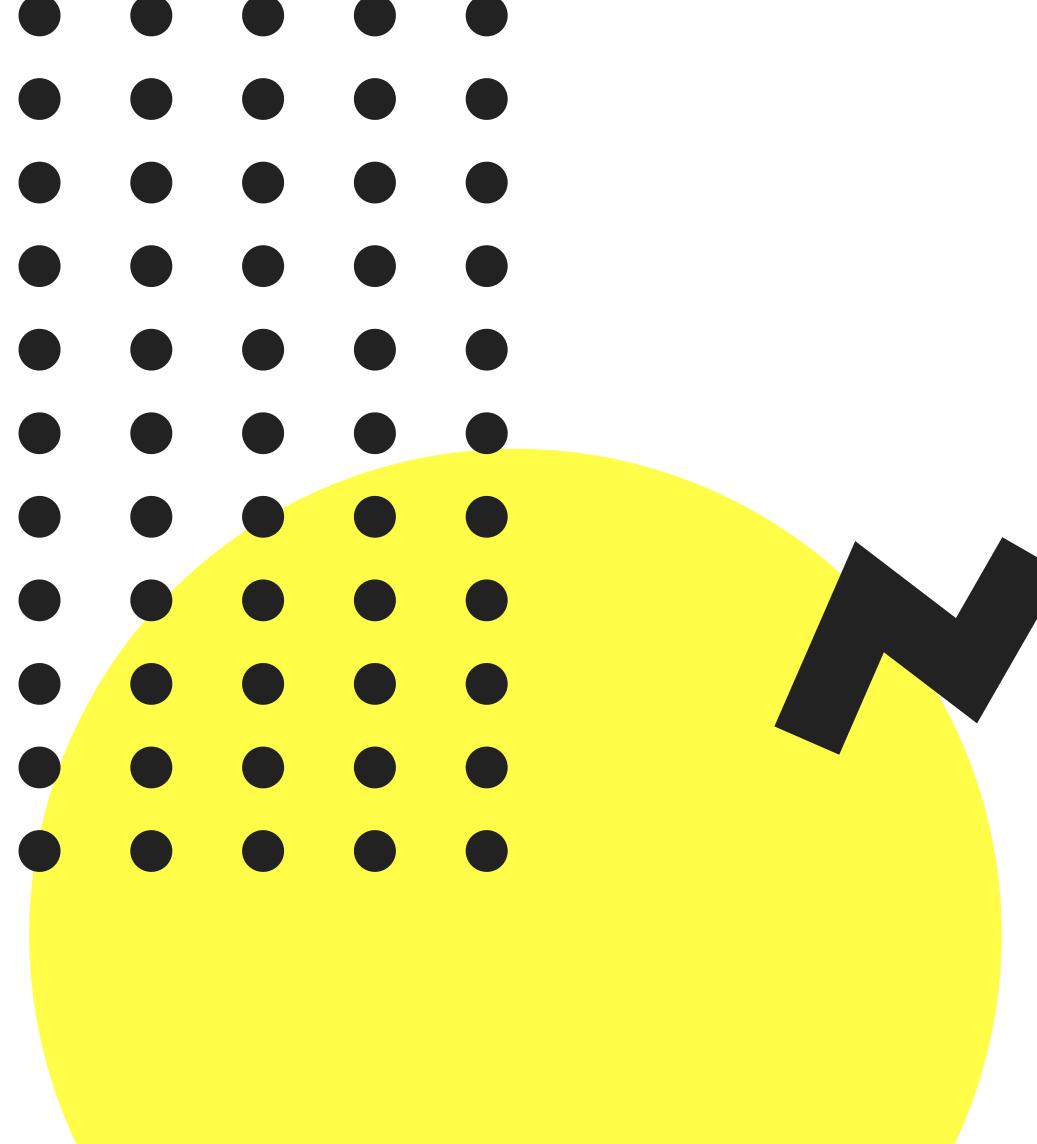
```
let bankBalance = 100;  
bankBalance += 200;  
bankBalance--;
```

# BOOLEANS

TRUE

or

FALSE



```
let isLoggedIn = true;  
let gameOver = false;  
const isWaterWet = true;
```

# Booleans

## TRUE OR FALSE

Booleans are very simple.  
You have two possible options: true  
or false. That's it!

# Variables Can Change Types



```
let numPuppies = 23; //Number  
numPuppies = false; //Now a Boolean  
numPuppies = 100; //Back to Number!
```

It doesn't really make sense to change from a number to a boolean here, but we can!



# Strings

## "STRINGS OF CHARACTERS"

Strings are another primitive type in JavaScript. They represent text, and must be wrapped in quotes.

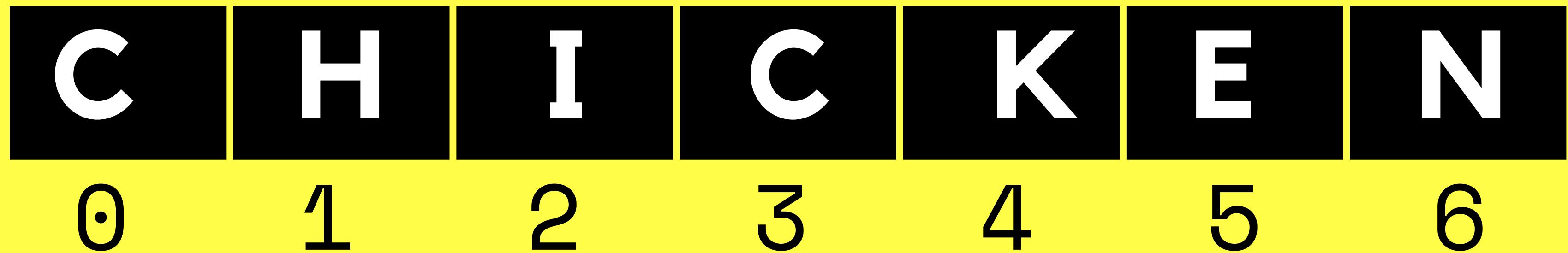
# STRINGS



```
let firstName = "Ziggy";           Double quotes work  
let msg = "Please do not feed the chimps!";  
let animal = 'Dumbo Octopus';      So do single quotes  
let bad = "this is wrong";        This DOES NOT work
```

It's fine to use either single or double quotes, just be consistent in your codebase.

# STRINGS ARE INDEXED



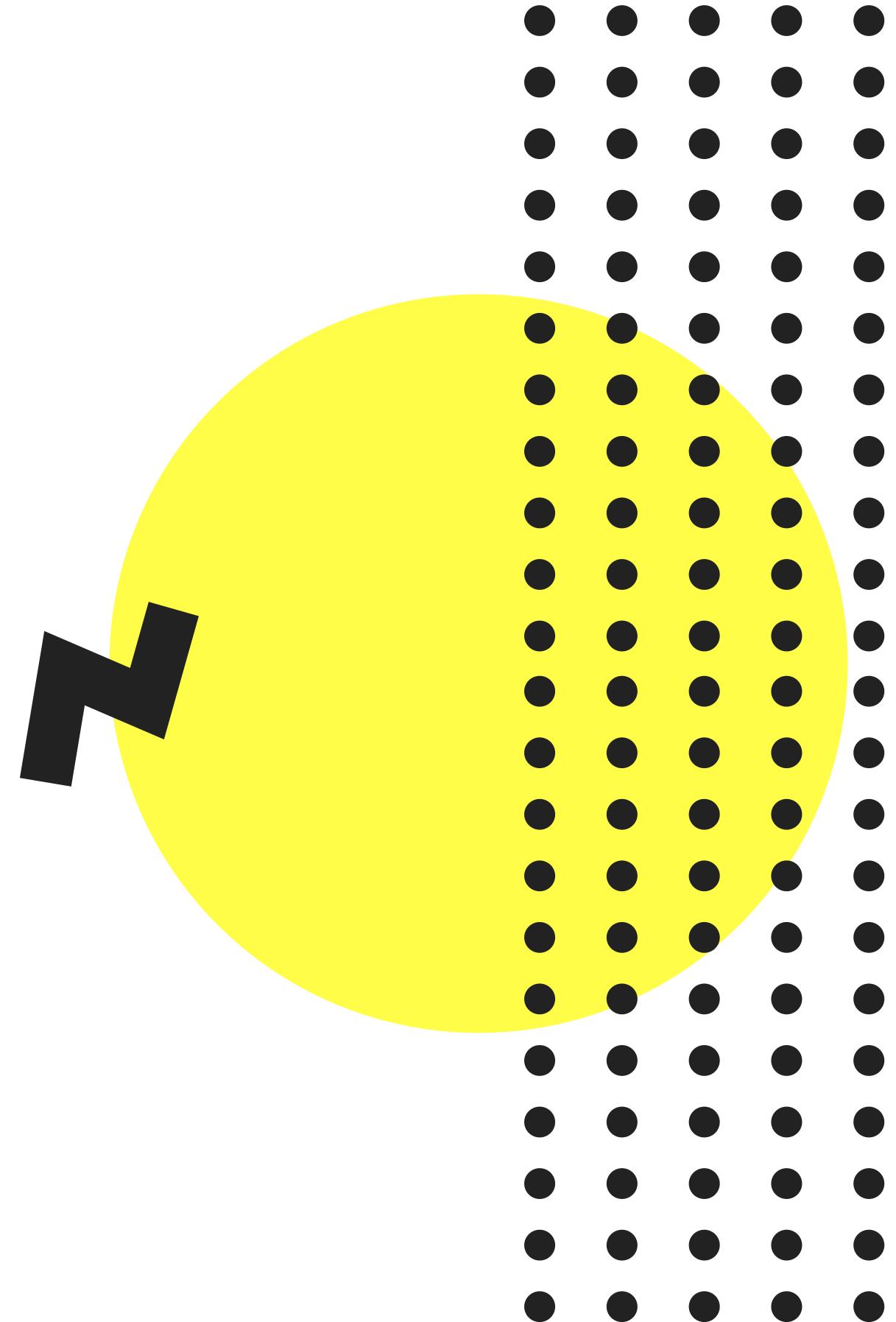
Each character has a corresponding index  
(a positional number)

# String Methods

**METHODS ARE BUILT-IN  
ACTIONS WE CAN PERFORM  
WITH INDIVIDUAL STRINGS**

They help us do things like:

- Searching within a string
- Replacing part of a string
- Changing the casing of a string



thing.method()

N

# Casing



```
let msg = 'I am king';
let yellMsg = msg.toUpperCase(); // 'I AM KING'

let angry = 'LeAvE mE aLoNe!';
angry.toLowerCase(); // 'leave me alone!'

//the value in angry is unchanged
angry; // 'LeAvE mE aLoNe!'
```

# Trim



```
let greeting = '  leave me alone plz  ';  
greeting.trim() // 'leave me alone plz'
```

# thing.method(arg)

Some methods accept **arguments** that modify their behavior.

Think of them as inputs that we can pass in.

We pass these arguments inside of the parentheses.

# indexOf



```
let tvShow = 'catdog';

tvShow.indexOf('cat'); // 0
tvShow.indexOf('dog'); // 3
tvShow.indexOf('z'); // -1 (not found)
```

# slice



```
let str = 'supercalifragilisticexpialidocious'  
  
str.slice(0,5); //'super'  
  
str.slice(5); // 'califragilisticexpialidocious'
```

# replace



```
let annoyingLaugh = 'teehee so funny! teehee!';

annoyingLaugh.replace('teehee', 'haha') // 'haha so funny! teehee!'
//Notice that it only replaces the first instance
```

# WHAT IS THE VALUE OF AGE?



```
const age = "5" + "4";
```

# WHAT DOES THIS EVALUATE TO?



"pecan pie"[7]

# WHAT DOES THIS EVALUATE TO?



"PUP"[3];

# What is the value of song?



```
let song = "london calling";  
song.toUpperCase();
```

# What is the value of *cleanedInput*?



```
let userInput = " T0DD@gmail.com";
let cleanedInput = userInput.trim().toLowerCase();
```

# What is the value of *index*?



```
let park = 'Yellowstone';
const index = park.indexOf('Stone');
```

# What is the value of *index*?



```
let yell = 'GO AWAY!!';
let index = yell.indexOf('!');
```

# WHAT DOES THIS EVALUATE TO?



```
'GARBAGE!'.slice(2).replace("B", "");
```

# STRING ESCAPES

- `\n` – newline
- `\'` – single quote
- `\"` – double quote
- `\\"` – backslash

# Template Literals

**SUPER USEFUL!**

```
```I counted ${3 + 4} sheep``; // "I counted 7 sheep"
```

TEMPLATE LITERALS ARE STRINGS THAT ALLOW EMBEDDED EXPRESSIONS, WHICH WILL BE EVALUATED AND THEN TURNED INTO A RESULTING STRING

**WE USE BACK-TICKS  
NOT SINGLE QUOTES**

``I am a template literal``

\* The back-tick key is usually above the tab key

# TEMPLATE LITERALS



```
let item = 'cucumbers';
let price = 1.99;
let quantity = 4;
```

```
`You bought ${quantity} ${item}, total price: ${price*quantity}`;
// "You bought 4 cucumbers, total price: $7.96"
```

# NULL & UNDEFINED

- Null
  - "Intentional absence of any value"
  - Must be assigned
- Undefined
  - Variables that do not have an assigned value are undefined

N

# NULL

```
1 // No one is logged in yet...
2 let loggedInUser = null; //value is explicitly nothing
3
4 // A user logs in...
5 loggedInUser = 'Alan Rickman';
```

# Undefined

```
1 let pickles; //We didn't assign a value
2 pickles; //undefined,
3 pickles = 'are very gross'
4
5 //Undefined also comes up in other situations:
6 let food = 'tacos';
7 food[7]; //undefined
```

# MATH OBJECT

Contains properties and methods for mathematical constants and functions



```
Math.PI // 3.141592653589793
```

```
//Rounding a number:  
Math.round(4.9) //5
```

```
//Absolute value:  
Math.abs(-456) //456
```

```
//Raises 2 to the 5th power:  
Math.pow(2,5) //32
```

```
//Removes decimal:  
Math.floor(3.9999) //3
```

# RANDOM NUMBERS

`Math.random()` gives us a random decimal between 0 and 1 (non-inclusive)



```
Math.random();  
//0.14502435424141957  
Math.random();  
//0.8937425043112937  
Math.random();  
//0.9759952148727442
```

# RANDOM INTEGERS

Let's generate random  
numbers between 1 and 10

```
const step1 = Math.random();  
//0.5961104892810127  
const step2 = step1 * 10  
//5.961104892810127  
const step3 = Math.floor(step2);  
//5  
const step4 = step3 + 1;  
//6  
  
Math.floor(Math.random() * 10) + 1;
```

# **parseInt & parseFloat**

Use to parse strings  
into numbers, but  
watch out for NaN!



```
parseInt('24') //24
parseInt('24.987') //24
parseInt('28dayslater') //28

parseFloat('24.987') //24.987
parseFloat('7') //7
parseFloat('i ate 3 shrimp') //NaN
```

Web Developer Bootcamp

# Boolean Logic

MAKING DECISIONS WITH JAVASCRIPT

# COMPARISONS

•

•

•

```
> // greater than  
< // less than  
>= // greater than or equal to  
<= // less than or equal to  
== // equality  
!= // not equal  
=== // strict equality  
!== // strict non-equality
```

# SOME EXAMPLES



```
10 > 1;      //true
0.2 > 0.3;   //false
-10 < 0;     //true
50.5 < 5;    //false
0.5 <= 0.5;  //true
99 >= 4;    //true
99 >= 99;   //true
'a' < 'b';   //true
'A' > 'a';  //false
```

Notice these all return a Boolean!

Though it's uncommon, you can compare strings. Just be careful, things get dicey when dealing with case, special characters, and accents!



VS



# `==` (double equals)

- Checks for equality of value, but not equality of type.
- It coerces both values to the same type and then compares them.
- This can lead to some unexpected results!

# `==` WEIRDNESS

```
● ● ●  
5 == 5;          //true  
'b' == 'c';    //false  
7 == '7';        //true  
0 == '';         //true  
true == false;  //false  
0 == false;     //true  
null == undefined; //true
```

# TRIPLE EQUALS

```
5 === 5; //true
1 === 2; //false
2 === '2'; //false
false === 0; //false

//Same applies for != and !==
10 != '10'; //false
10 !== '10'; //true
```

CHECKS FOR EQUALITY OF VALUE AND TYPE

# console.log()

prints arguments to the console

(we need this if we're going to start working with files!)



# Running Code From a File

**app.js**

```
● ● ●  
//Put your code in the JS File  
alert('Hello from JS!');  
  
//Won't show up!!  
"hi".toUpperCase();  
  
//Will show up!  
console.log("hi".toUpperCase());
```

**demo.html**

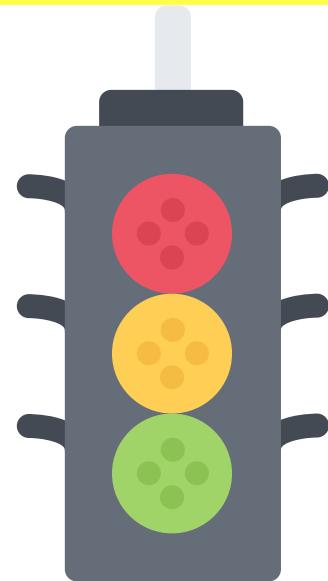
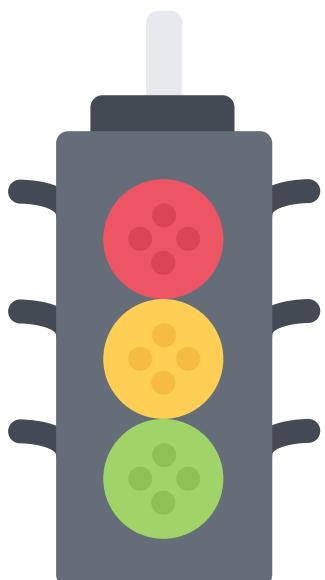
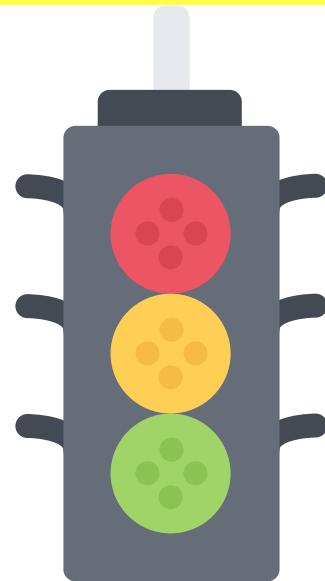
```
● ● ●  
<!DOCTYPE html>  
<html>  
<head>  
  <title>JS Demo</title>  
  <script src="app.js"></script>  
</head>  
<body>  
  
</body>  
</html>
```

**Write your code  
in a .js file**

**Include your script  
in a .html file**

# Conditionals

**MAKING DECISIONS WITH CODE**



# IF STATEMENT

Only runs code if given condition is true



```
let rating = 3;  
  
if (rating === 3) {  
  console.log("YOU ARE A SUPERSTAR!");  
}
```

# ELSE IF

If not the first thing, maybe this other thing??

```
let rating = 2;

if (rating === 3) {
  console.log("YOU ARE A SUPERSTAR!");
}
else if (rating === 2) {
  console.log("MEETS EXPECTATIONS");
}
```

# ELSE IF

We can add multiple else ifs!

```
let rating = 1;

if (rating === 3) {
  console.log("YOU ARE A SUPERSTAR!");
}
else if (rating === 2) {
  console.log("MEETS EXPECTATIONS");
}
else if (rating === 1) {
  console.log("NEEDS IMPROVEMENT");
}
```

# ELSE

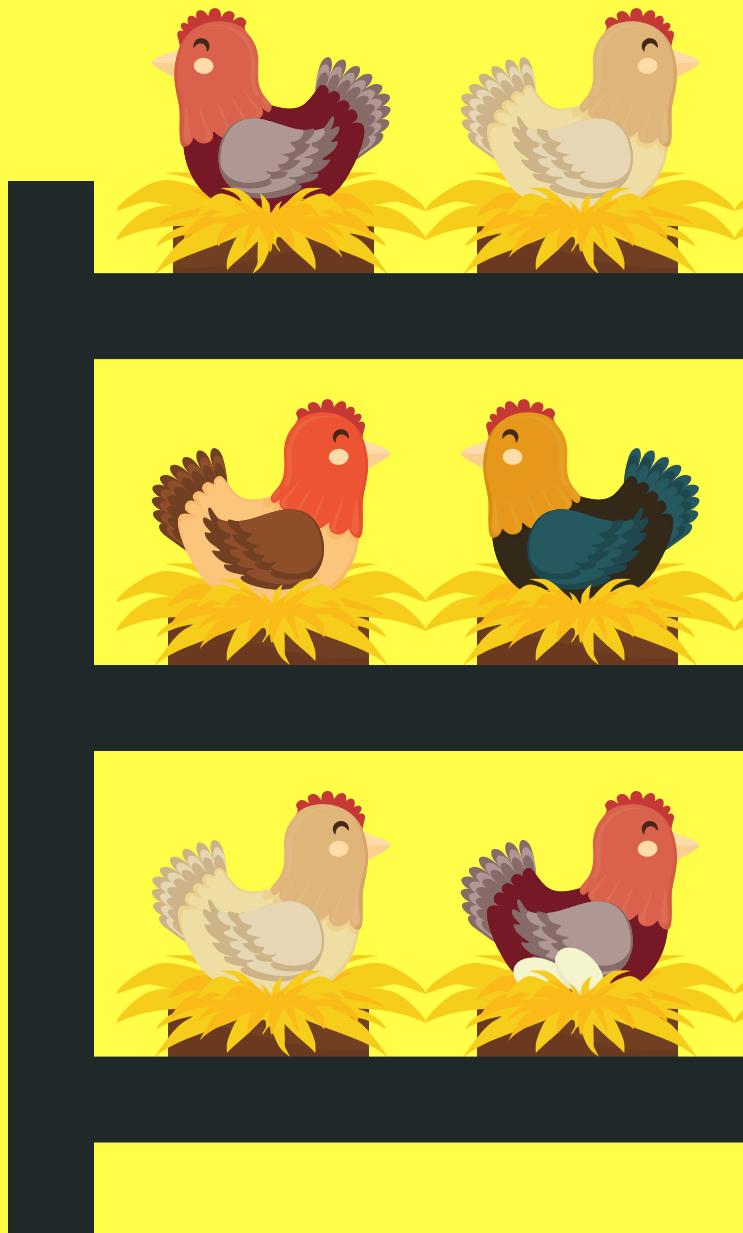
If nothing else was true, do this...

```
● ● ●  
  
let rating = -99;  
  
if (rating === 3) {  
    console.log("YOU ARE A SUPERSTAR!");  
}  
else if (rating === 2) {  
    console.log("MEETS EXPECTATIONS");  
}  
else if (rating === 1) {  
    console.log("NEEDS IMPROVEMENT");  
}  
else {  
    console.log("INVALID RATING!");  
}
```

# NESTING

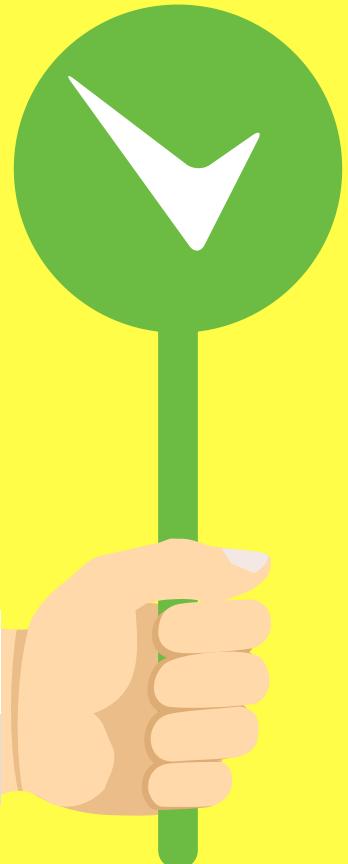
We can nest conditionals inside conditionals

```
let password = "cat dog";
if (password.length >= 6) {
  if (password.indexOf(' ') !== -1) {
    console.log("Password cannot include spaces");
  }
  else {
    console.log("Valid password!!")
  }
}
else {
  console.log("Password too short!");
}
```



# TRUTHY AND FALSEY VALUES

- All JS values have an inherent truthyness or falsyness about them
- Falsy values:
  - false
  - 0
  - "" (empty string)
  - null
  - undefined
  - NaN
- Everything else is truthy!



# Logical Operators

COMBINING EXPRESSIONS

& &

||

!

# AND

Both sides must be true, for the entire thing to be true



```
1 <= 4 && 'a' === 'a'; //true
```

```
9 > 10 && 9 >= 9; //false
```

```
'abc'.length === 3 && 1+1 === 4; //false
```

# AND

Both sides must be true, for the entire thing to be true

```
let password = 'taco tuesday';

if(password.length >= 6 && password.indexOf(' ') === -1){
    console.log("Valid Password!");
}
else {
    console.log("INVALID PASSWORD!");
}
```

# OR

If one side is true, the entire thing is true

```
//only one side needs to be true!  
1 !== 1 || 10 === 10 //true  
10/2 === 5 || null //true  
0 || undefined //false
```



OR

If one side is true, the entire thing is true



```
let age = 76;

if(age < 6 || age >= 65){
    console.log('You get in for free!');
}
else {
    console.log('That will be $10 please');
}
```

# NOT

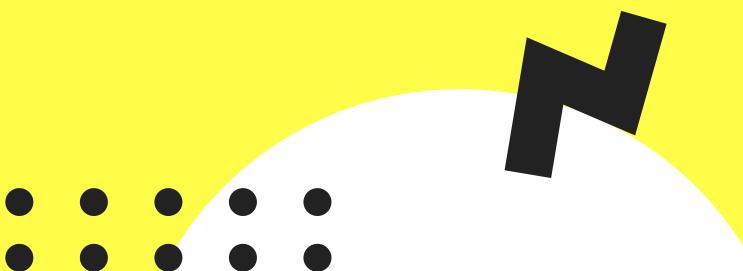
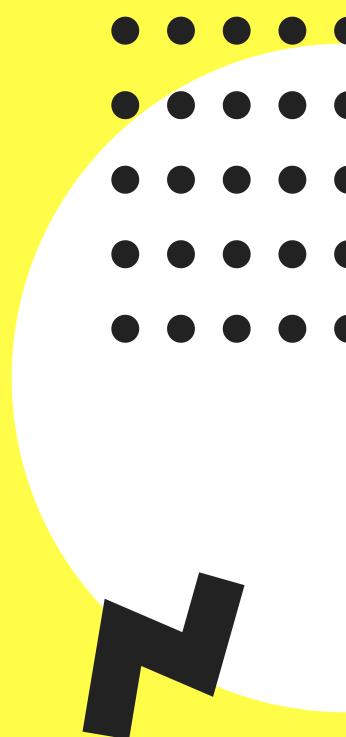
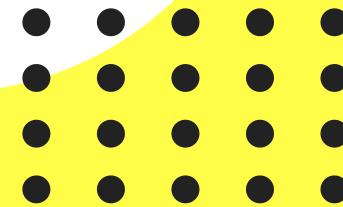
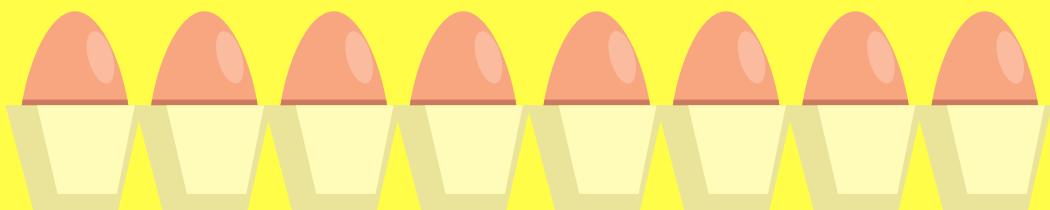
`!expression` returns true if expression is false

```
!null //true  
  
!(0 === 0) //false  
  
!(3 <= 4) //false
```

Web Developer Bootcamp

# JS Arrays

OUR FIRST DATA STRUCTURE



# ARRAYS

Ordered collections of values.

- List of comments on IG post
- Collection of levels in a game
- Songs in a playlist



# Creating Arrays

```
● ● ●  
// To make an empty array  
let students = [];  
  
//An array of strings  
let colors = ['red', 'orange', 'yellow'];  
  
//An array of numbers  
let lottoNums = [19,22,56,12,51];  
  
//A mixed array  
let stuff = [true, 68, 'cat', null];
```

# ARRAYS ARE INDEXED



Each element has a corresponding index  
(counting starts at 0)

# Arrays Are Indexed

```
● ● ● ●  
let colors = ['red', 'orange', 'yellow', 'green'];  
  
colors.length //4  
  
colors[0] //'red'  
colors[1] //'orange'  
colors[2] //'yellow'  
colors[3] //'green'  
colors[4] //'undefined'
```

# Modifying Arrays

```
● ● ● ●  
let colors = ['rad','orange','green','yellow'];  
  
colors[0] = 'red';  
  
colors[2] = 'yellow';  
colors[3] = 'green';  
  
colors[4]; //undefined  
colors[4] = 'blue';  
//["red", "orange", "yellow", "green", "blue"]
```

# ARRAY METHODS

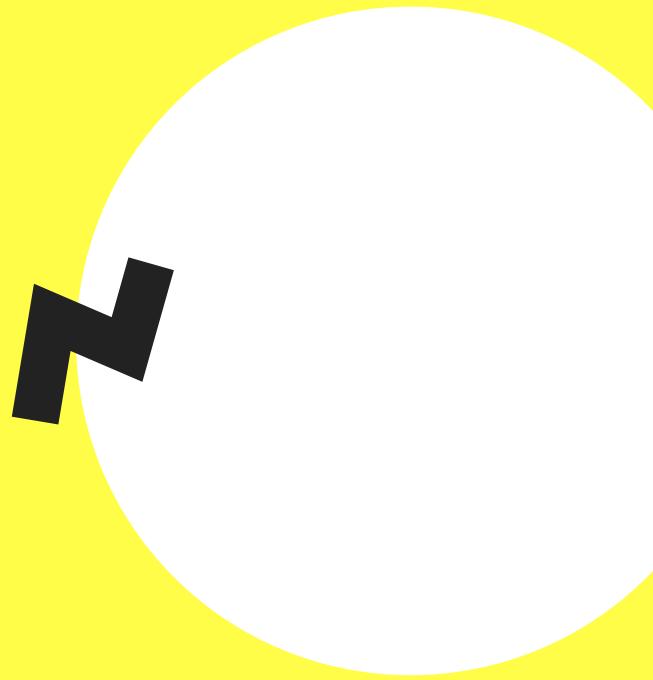
**Push** – add to end

**Pop** – remove from end

**Shift** – remove from start

**Unshift** – add to start

YOU'LL GET USED TO THE NAMES EVENTUALLY!



# MORE METHODS

**concat** - merge arrays

**includes** - look for a value

**indexOf** - just like string.indexOf

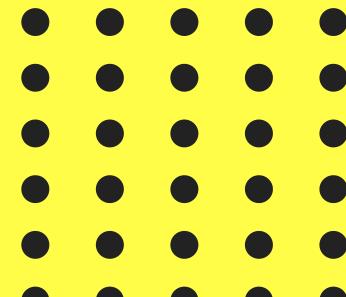
**join** - creates a string from an array

**reverse** - reverses an array

**slice** - copies a portion on an array

**splice** - removes/replaces elements

**sort** - sorts an array

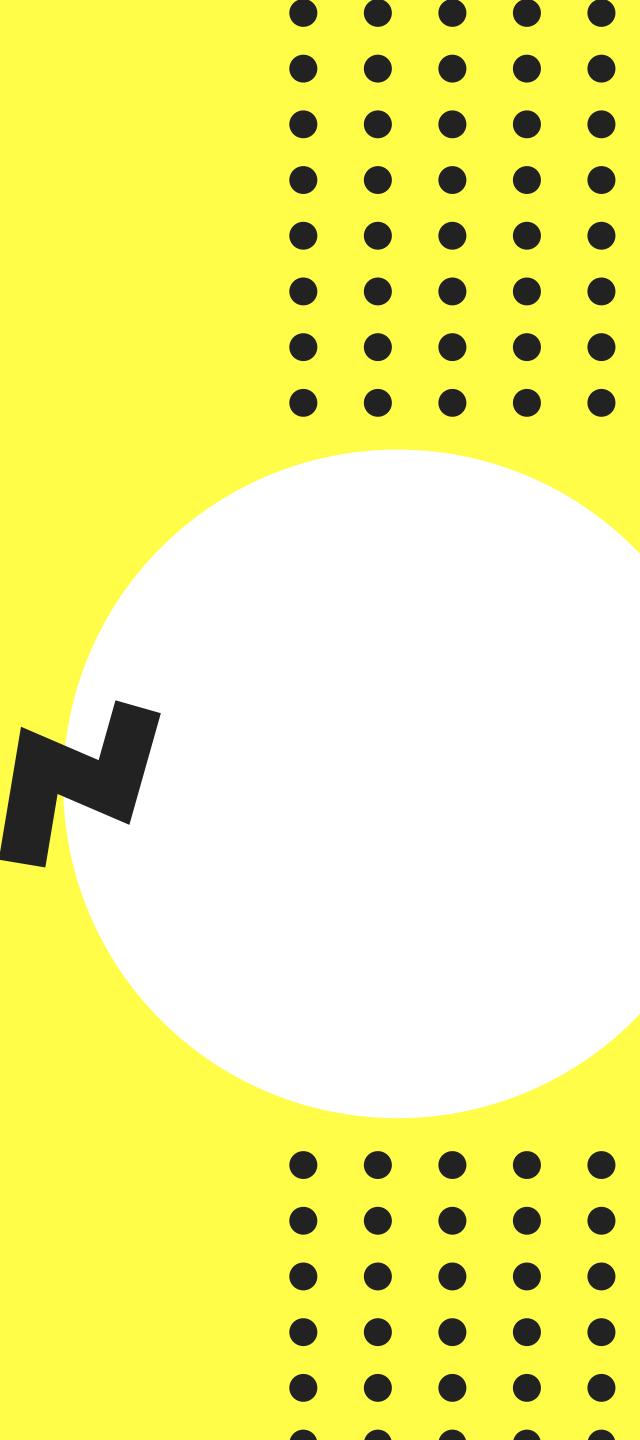


CONST

AND

ARRAYS

WHY DO PEOPLE USE CONST WITH ARRAYS??



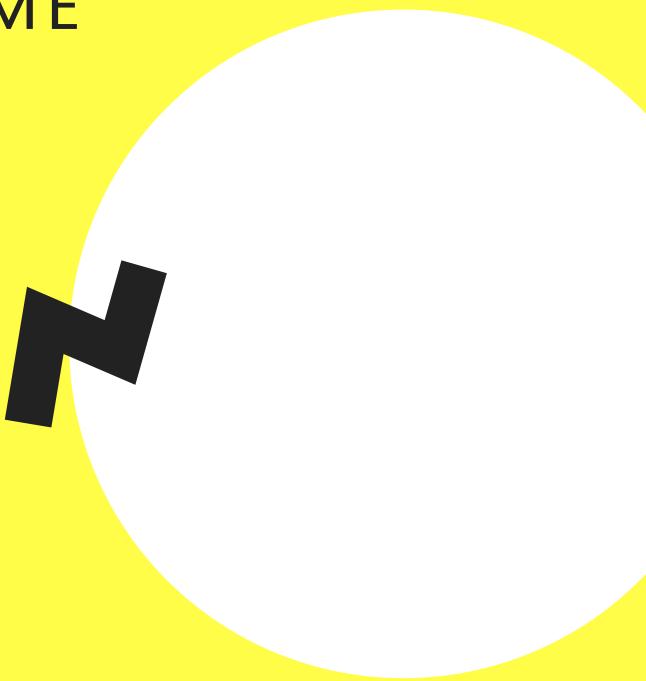
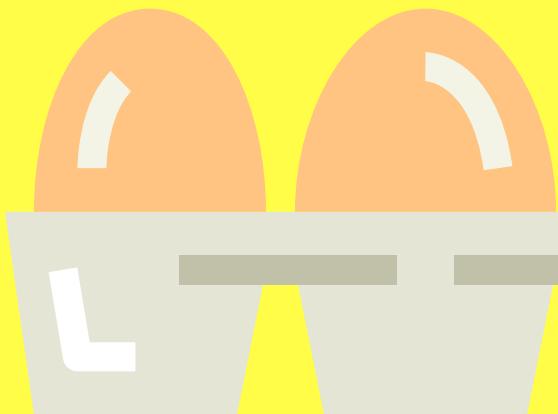
# THE VALUES CAN CHANGE

AS LONG AS THE REFERENCE REMAINS THE SAME



```
const myEggs = ['brown', 'brown'];
```

myEggs →



# THE VALUES CAN CHANGE

AS LONG AS THE REFERENCE REMAINS THE SAME



```
const myEggs = ['brown', 'brown'];
myEggs.push('purple');
```

myEggs →



# THE VALUES CAN CHANGE

AS LONG AS THE REFERENCE REMAINS THE SAME



```
const myEggs = ['brown', 'brown'];
myEggs.push('purple');
myEggs[0] = 'green';
```

myEggs →



# THE VALUES CAN CHANGE

AS LONG AS THE REFERENCE REMAINS THE SAME



```
const myEggs = ['brown', 'brown'];
myEggs.push('purple');
myEggs[0] = 'green';

myEggs = ['blue', 'pink']; //NO!
```

myEggs →

✖ ▶Uncaught TypeError: Assignment  
to constant variable.



# NESTED ARRAYS

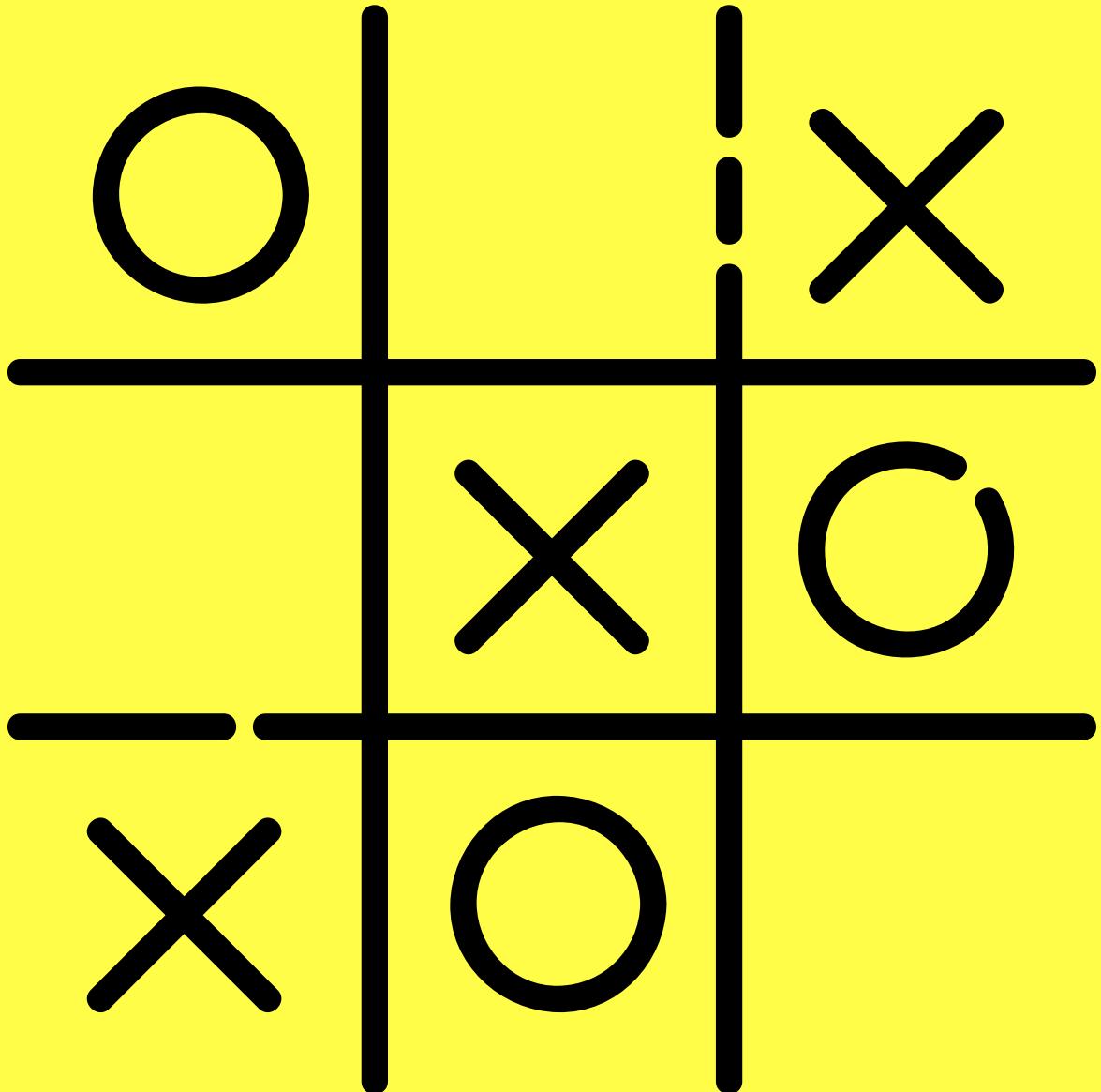
We can store arrays inside other arrays!



```
const colors = [
  ['red', 'crimson'],
  ['orange', 'dark orange'],
  ['yellow', 'golden rod'],
  ['green', 'olive'],
  ['blue', 'navy blue'],
  ['purple', 'orchid']
]
```

# NESTED ARRAYS

```
const board = [  
    ['0', null, 'X'],  
    [null, 'X', '0'],  
    ['X', '0', null]  
]
```



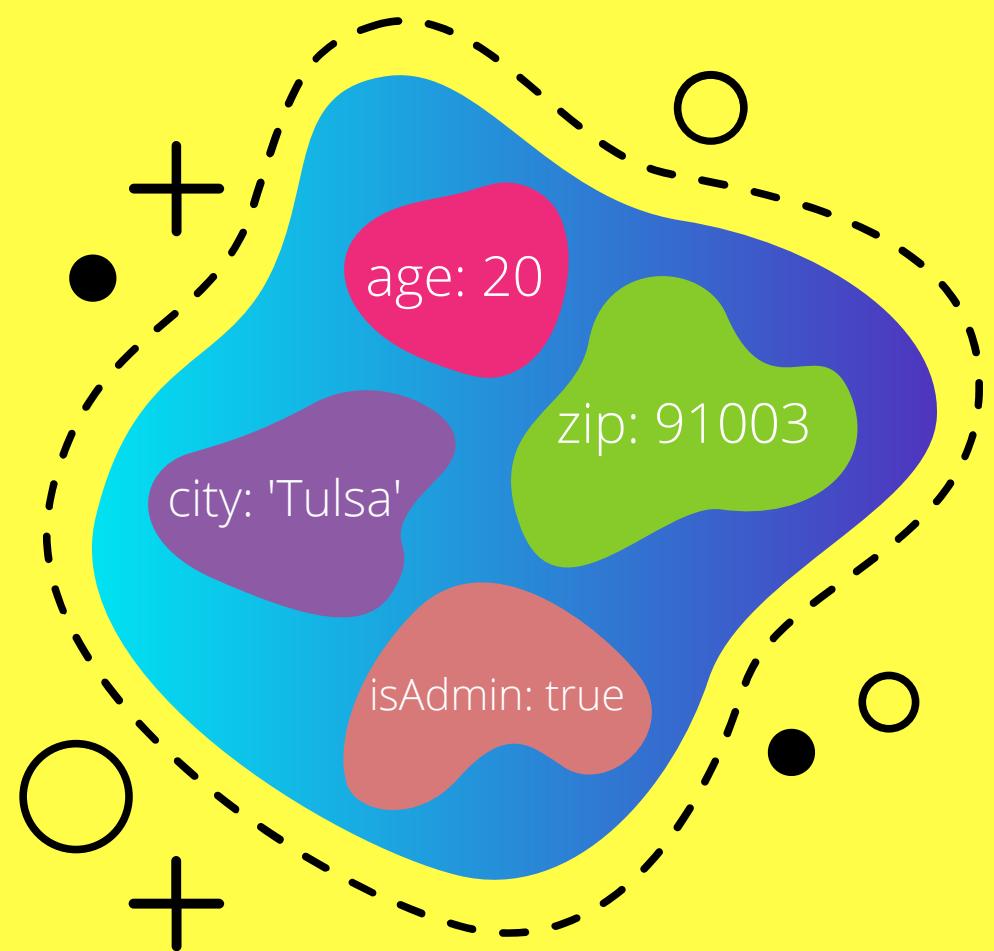
Web Developer Bootcamp

# JS Objects

OUR SECOND DATA STRUCTURE!

# OBJECTS

- Objects are collections of properties.
- Properties are a key-value pair
- Rather than accessing data using an index, we use custom keys.



# HOW WOULD YOU STORE THIS?



# Using an Object!

```
● ● ●  
const fitBitData = {  
    totalSteps : 308727,  
    totalMiles : 211.7,  
    avgCalorieBurn : 5755,  
    workoutsThisWeek : '5 of 7',  
    avgGoodSleep : '2:13'  
};
```

**PROPERTY =**

**KEY**

**+**

**VALUE**



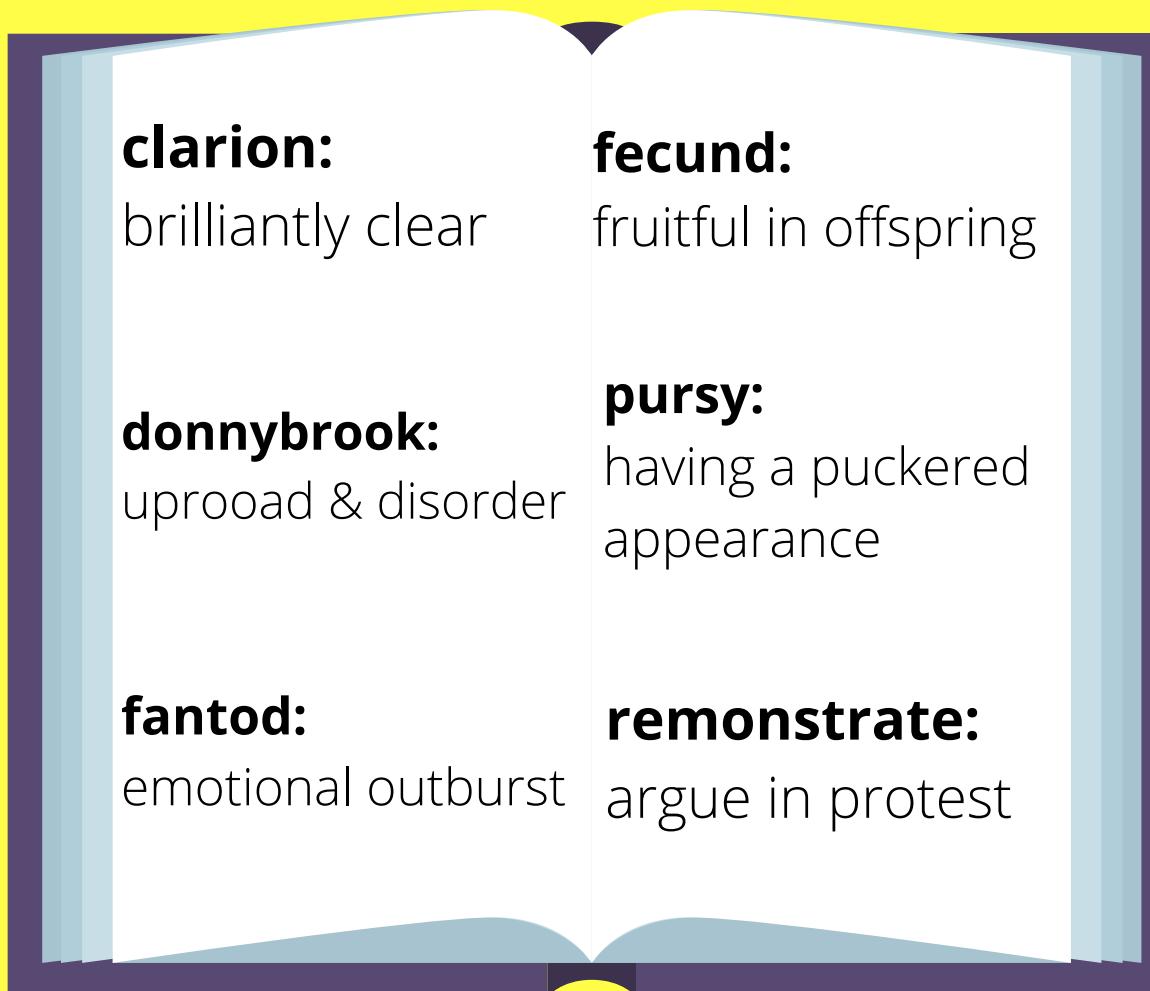
# KEY-VALUE PAIRS

username: → 'crazyCatLady'

upvotes: → 7

text → 'great post!'

# DICTIONARY



# ALL TYPES WELCOME!

```
let comment = {  
    username      : 'sillyGoose420',  
    downVotes     : 19,  
    upVotes       : 214,  
    netScore      : 195,  
    commentText   : 'Tastes like chicken lol',  
    tags: ['#hilarious', '#funny', '#silly'],  
    isGilded: false  
};
```

# VALID KEYS

All keys are  
converted to  
strings \*

\* Except for Symbols, which we haven't covered yet



# ACCESSING DATA



```
const palette = {  
  red: '#eb4d4b',  
  yellow: '#f9ca24',  
  blue: '#30336b'  
}
```



```
palette.red // "#eb4d4b"
```

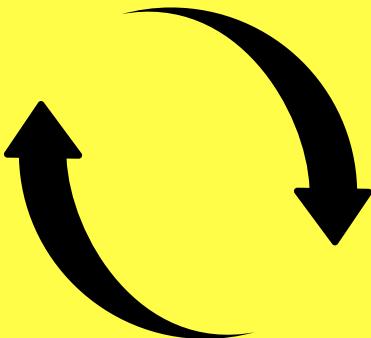


```
palette['blue'] // "#30336b"
```



```
let color = 'yellow';  
palette[color] // "#f9ca24"
```

# UPDATING & ADDING PROPERTIES



```
const fitBitData = {  
    totalSteps      : 308727,  
    totalMiles     : 211.7,  
    avgCalorieBurn : 5755,  
    workoutsThisWeek: '5 of 7',  
    avgGoodSleep   : '2:13'  
};  
//Updating properties:  
fitBitData.workoutsThisWeek = '6 of 7';  
fitBitData.totalMiles += 7.5;  
  
//Adding a new property  
fitBitData.heartStillBeating = true;
```

# ARRAYS + OBJECTS

```
const shoppingCart = [  
  {  
    product: 'Jenga Classic',  
    price: 6.88,  
    quantity: 1  
  },  
  {  
    product: 'Echo Dot',  
    price: 29.99,  
    quantity: 3  
  },  
  {  
    product: 'Fire Stick',  
    price: 39.99,  
    quantity: 2  
  }  
]
```

```
const student = {  
  firstName: 'David',  
  lastName: 'Jones',  
  strengths: ['Music', 'Art'],  
  exams: {  
    midterm: 92,  
    final: 88  
  }  
}
```

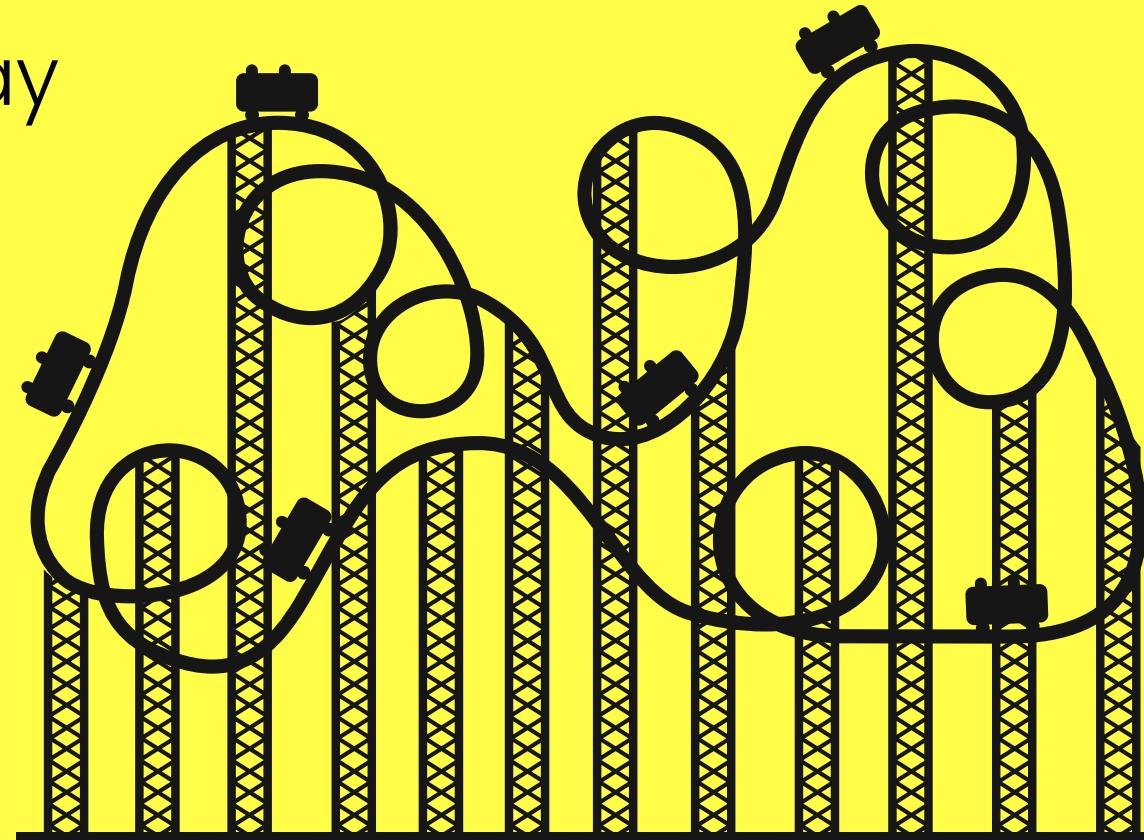
The Web Developer Bootcamp

# Js Loops

REPEAT STUFF. REPEAT STUFF. REPEAT STUFF.

# LOOPS

- Loops allow us to repeat code
  - "Print 'hello' 10 times
  - Sum all numbers in an array
- There are multiple types:
  - for loop
  - while loop
  - for...of loop
  - for...in loop



For  
Loops  
Buckle Up!



N

N



# For Loop Syntax

```
for (  
    [initialExpression];  
    [condition];  
    [incrementExpression]  
)
```

# Our First For Loop



start at 1

stop at 10

add 1 each time

```
for (let i = 1; i <= 10; i++) {  
    console.log(i);  
}
```

# Another Example

```
● ● ●  
for (let i = 50; i >= 0; i -= 10) {  
    console.log(i);  
}  
//50  
//40  
//30  
//20  
//10  
//0
```

- Start *i* at 50
- Subtract 10 each iteration
- Keep going as long as *i*  $\geq 0$

# Infinite Loops

```
● ● ●  
//DO NOT RUN THIS CODE!  
for (let i = 20; i >= 0; i++) {  
    console.log(i);  
} //BADDADDD!!!
```



# Looping Over Arrays

```
const animals = [ 'lions', 'tigers', 'bears' ];  
  
for (let i = 0; i < animals.length; i++) {  
  console.log(i, animals[i]);  
}  
//0 'lions'  
//1 'tigers'  
//2 'bears'
```

To loop over an array, start at index 0 and continue looping to until last index (`length-1`)

# NESTED LOOPS



```
let str = 'LOL';
for (let i = 0; i <= 4; i++) {
  console.log("Outer:", i);
  for (let j = 0; j < str.length; j++) {
    console.log('  Inner:', str[j]);
  }
}
```

```
Outer: 0
Inner: L
Inner: 0
Inner: L

Outer: 1
Inner: L
Inner: 0
Inner: L

Outer: 2
Inner: L
Inner: 0
Inner: L

Outer: 3
Inner: L
Inner: 0
Inner: L

Outer: 4
Inner: L
Inner: 0
Inner: L
```

# While Loops



```
let num = 0;  
while (num < 10) {  
    console.log(num);  
    num++;  
}
```

0

1

2

3

4

While loops continue running as long as the test condition is true.

# A Common Pattern

```
● ● ●  
let targetNum = Math.floor(Math.random() * 10);  
let guess = Math.floor(Math.random() * 10);  
  
while (guess !== targetNum) {  
    console.log(`Guessed ${guess}...Incorrect!`);  
    guess = Math.floor(Math.random() * 10);  
}  
console.log(`CORRECT! Guessed: ${guess} & target was: ${targetNum}`);
```

# The Break Keyword



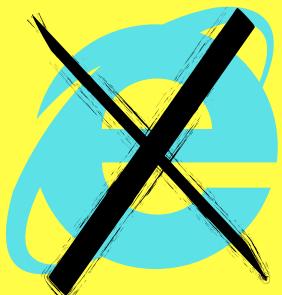
```
let targetNum = Math.floor(Math.random() * 10);
let guess = Math.floor(Math.random() * 10);

while (true) {
    guess = Math.floor(Math.random() * 10);
    if (guess === targetNum) {
        console.log(`CORRECT! Guessed: ${guess} & target was: ${targetNum}`);
        break;
    }
    else {
        console.log(`Guessed ${guess}...Incorrect!`);
    }
}
```

# FOR...OF

A nice and easy way of  
iterating over arrays

(or other iterable objects)



No Internet  
Explorer Support

# For...Of

```
for (variable of iterable) {  
    statement  
}
```

# An Example

```
let subreddits = [ 'soccer', 'popheads', 'cringe', 'books' ];
for (let sub of subreddits) {
    //Do this for every item in subreddits array:
    console.log(` ${sub} - www.reddit.com/r/${sub}`);
}
```

# Nested For...Of

```
const magicSquare = [
  [ 2, 7, 6 ],
  [ 9, 5, 1 ],
  [ 4, 3, 8 ]
];

for (let row of magicSquare) {
  let sum = 0;
  for (let num of row) {
    sum += num;
  }
  console.log(`Row of ${row} sums to ${sum}`);
}
```

Web Developer Bootcamp

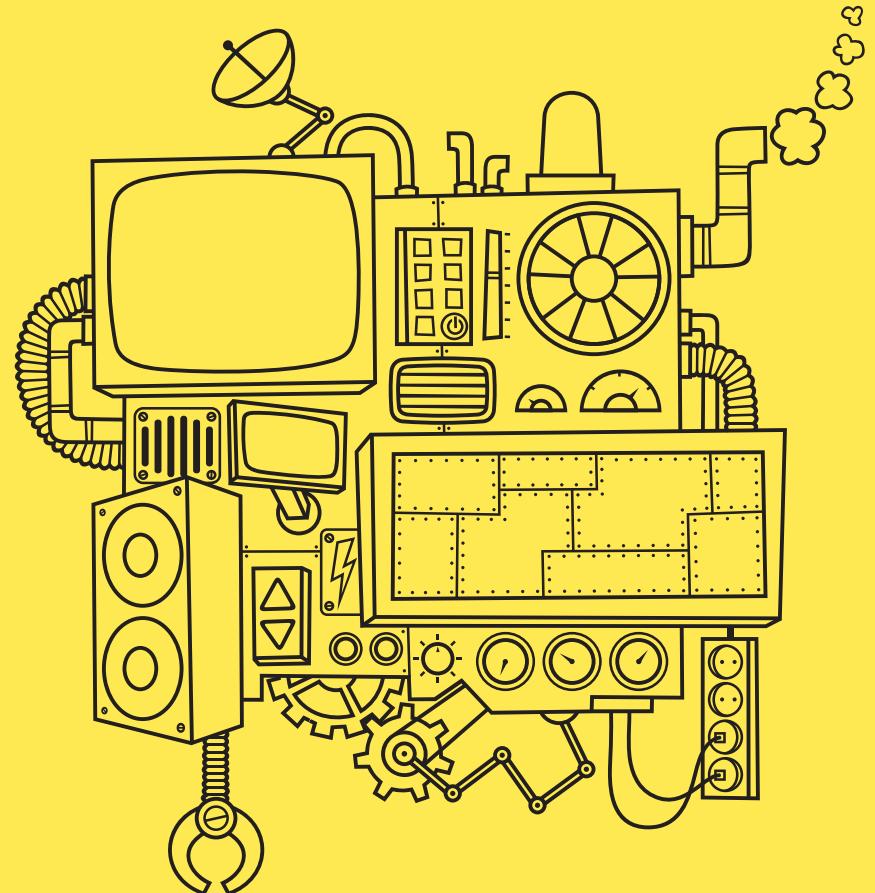
# JavaScript Functions

THE LAST "BIG" TOPIC!

# FUNCTIONS

Reusable procedures

- Functions allow us to write reusable, modular code
- We define a "chunk" of code that we can then execute at a later point.
- We use them ALL THE TIME



# 2 STEP PROCESS

DEFINE



RUN



# DEFINE

```
function funcName() {  
    //do something  
}
```

# DEFINE



```
function grumpus() {  
    console.log('ugh...you again...');  
    console.log('for the last time...');  
    console.log('LEAVE ME ALONE!!!!');  
}
```

# RUN

```
funcName(); //run once
```

```
funcName(); //run again!
```

# RUN



```
grumpus();  
//ugh...you again...  
//for the last time...  
//LEAVE ME ALONE!!!
```

```
grumpus();  
//ugh...you again...  
//for the last time...  
//LEAVE ME ALONE!!!
```



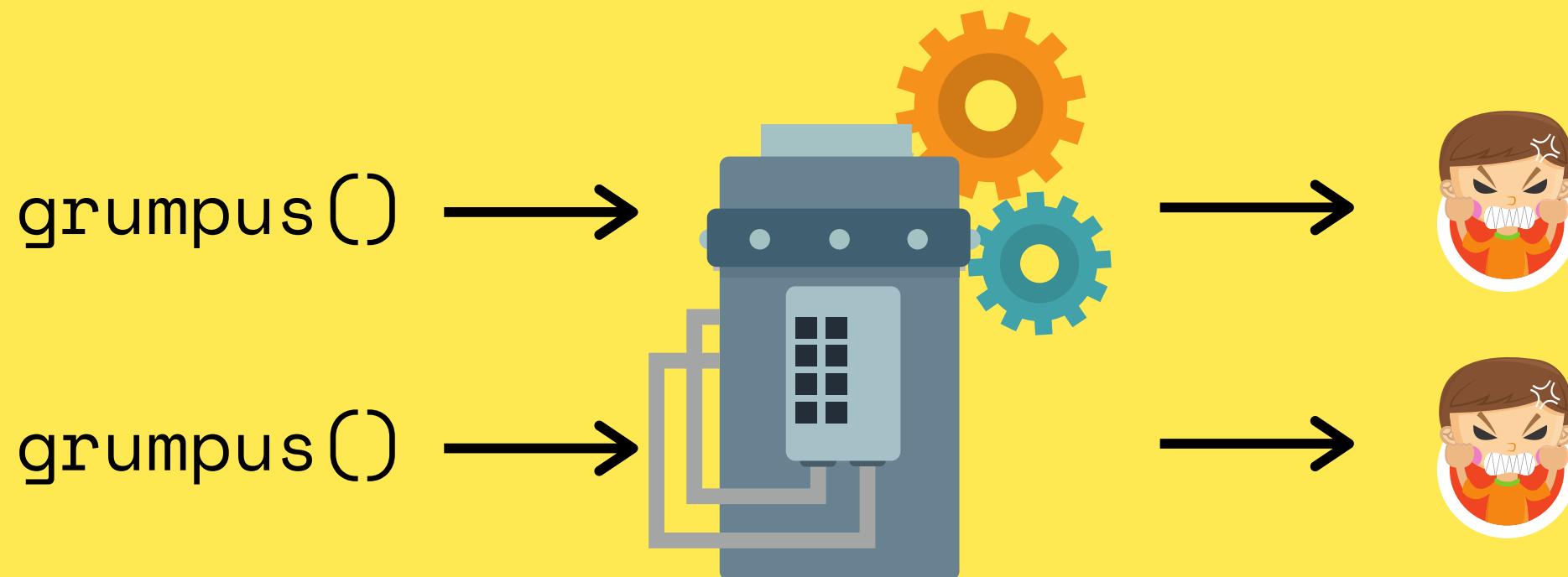


# ARGUMENTS

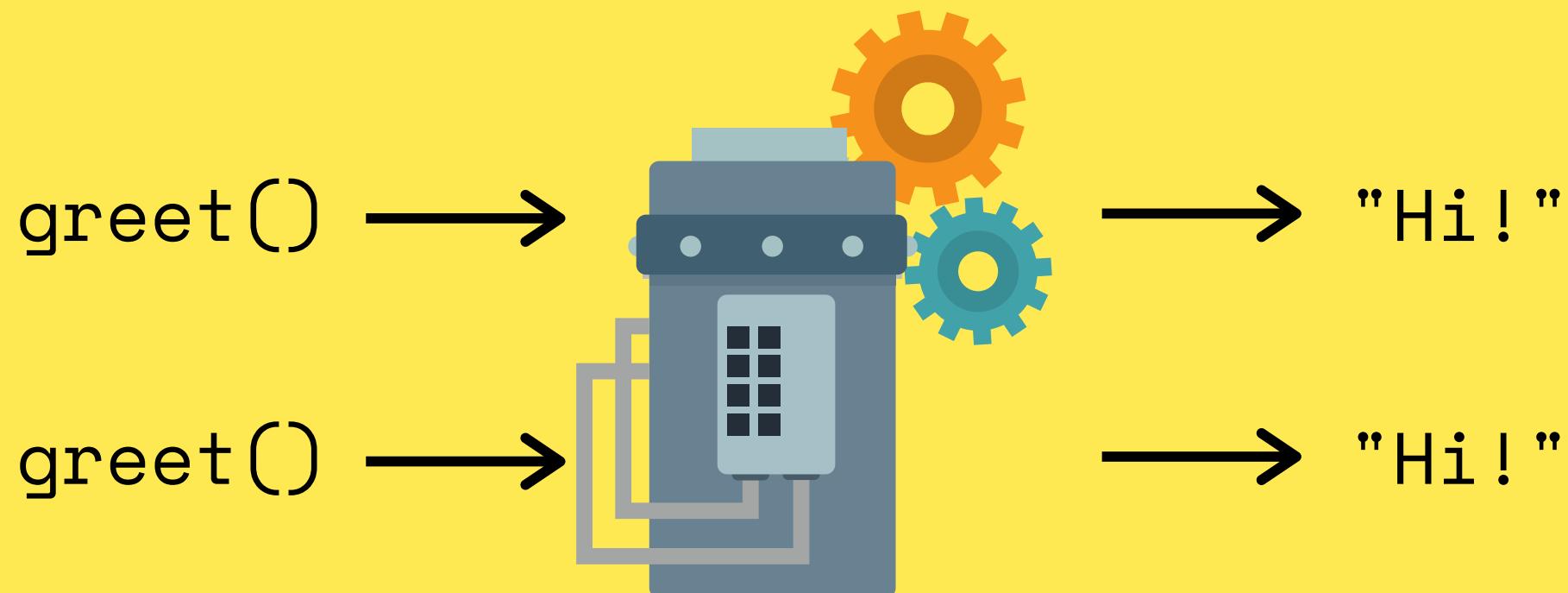
# INPUTS

Right now, our simple functions accept zero inputs. They behave the same way every time.

# NO INPUTS



# NO INPUTS

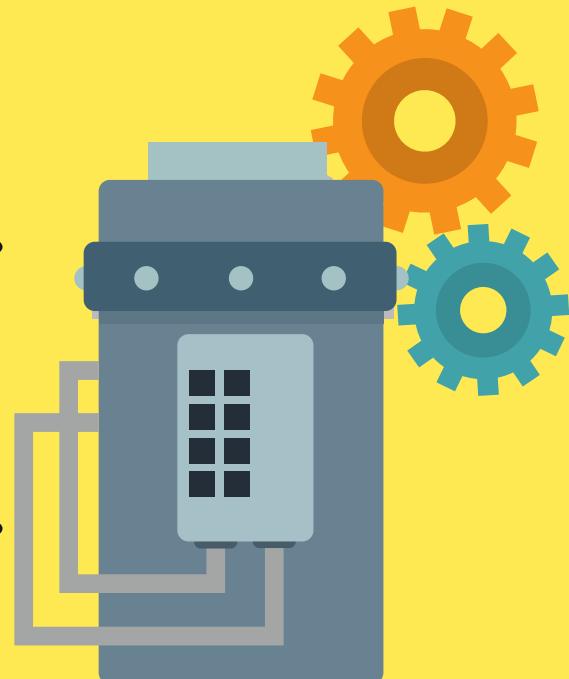


# ARGUMENTS

We can also write functions that accept inputs, called arguments

# ARGUMENTS

greet('Tim') →

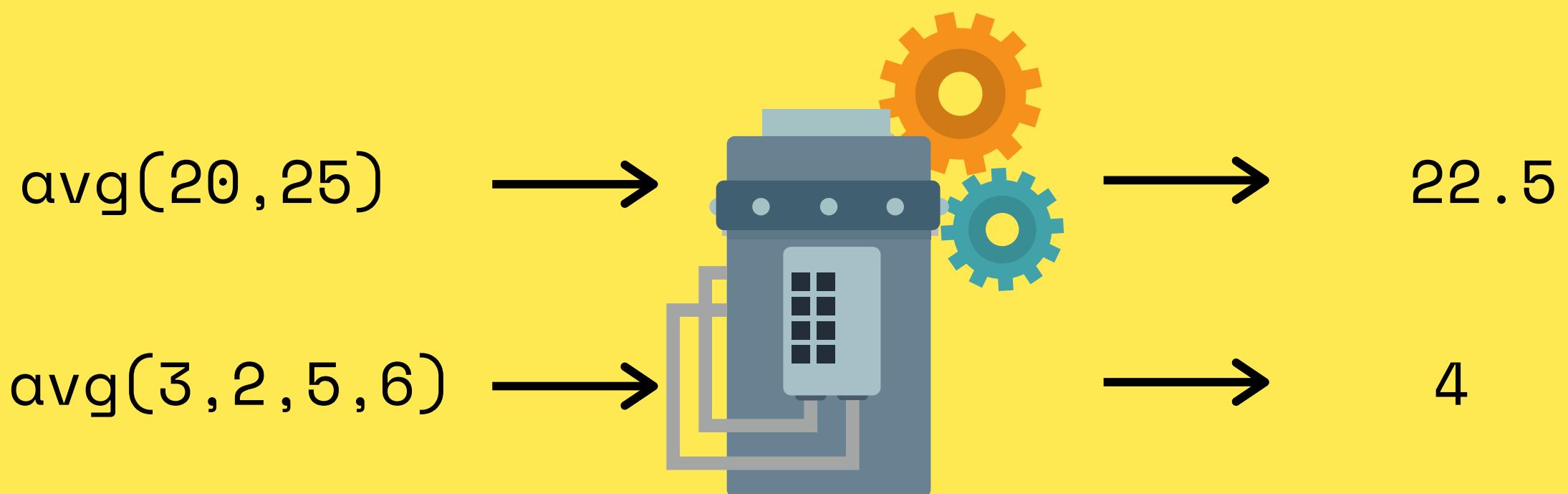


→ "Hi Tim!"

greet('Anya') →

→ "Hi Anya!"

# ONE MORE



# We've seen this before

No inputs

```
//No input  
"hello".toUpperCase();
```

Arguments!

```
//Different inputs...  
"hello".indexOf('h'); //0  
//Different outputs...  
"hello".indexOf('o'); //4
```

# GREET TAKE 2



```
function greet(person) {  
  console.log(`Hi, ${person}!`);  
}
```



```
greet('Arya');
```

→ "Hi, Arya!"



```
greet('Ned');
```

→ "Hi, Ned!"

# 2 ARGUMENTS!



```
function findLargest(x, y) {  
    if (x > y) {  
        console.log(`${x} is larger!`);  
    }  
    else if (x < y) {  
        console.log(`${y} is larger!`);  
    }  
    else {  
        console.log(`${x} and ${y} are equal!`);  
    }  
}
```



```
findLargest(-2,77)
```

"77 is  
larger!"



```
findLargest(33,33);
```

"33 and 33  
are equal"

# RETURN

Built-in methods `return` values  
when we call them.  
We can store those values:



```
const yell = "I will end you".toUpperCase();

yell; // "I WILL END YOU"

const idx = ['a', 'b', 'c'].indexOf('c');

idx; // 2
```

# NO RETURN!

Our functions print values out, but do  
NOT return anything



```
● ● ●  
  
function add(x, y) {  
  console.log(x + y);  
}  
  
const sum = add(10, 16);  
sum; //undefined
```



# FIRST RETURN!

Now we can capture a return value in a variable!



```
function add(x, y) {  
    return x + y; //RETURN!  
}  
  
const sum = add(10, 16);  
sum; //26  
  
const answer = add(100, 200);  
answer; //300
```

A dark rectangular box containing a snippet of JavaScript code. The code defines a function named 'add' that takes two parameters, 'x' and 'y', and returns their sum. It then shows two examples of calling this function: first with arguments 10 and 16, resulting in the output 26; and second with arguments 100 and 200, resulting in the output 300. The word 'RETURN!' is written in all caps inside the code block.

# RETURN

The return statement ends function execution AND specifies the value to be returned by that function

# FUNCTIONS IN DETAIL

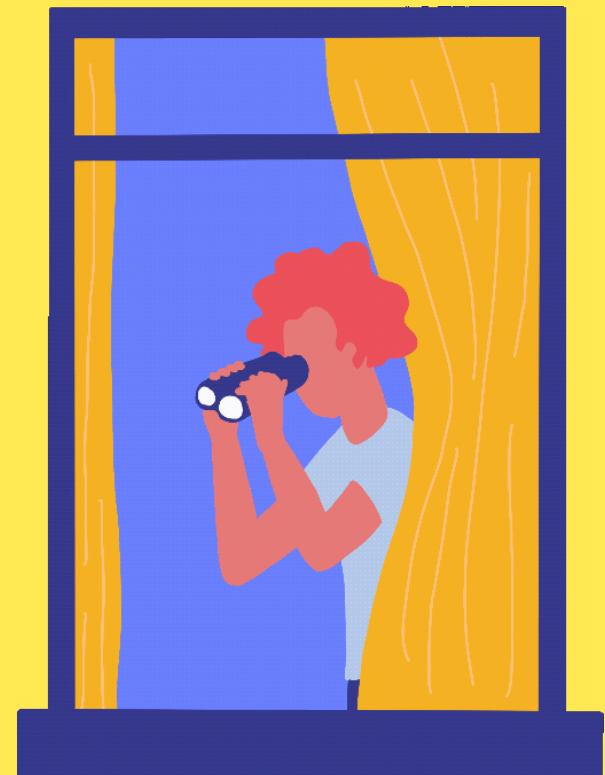
Important things  
you should know  
about functions



# SCOPE

## Variable "visibility"

The location where a variable is defined dictates where we have access to that variable



# FUNCTION SCOPE

```
function helpMe( ){  
  
    let msg = "I'm on fire!";  
  
    msg; // "I'm on fire";  
}  
  
msg; // NOT DEFINED!
```

*msg* is scoped to the  
*helpMe* function

# FUNCTION SCOPE

```
let bird = 'mandarin duck';

function birdWatch(){
    let bird = 'golden pheasant';
    bird; // 'golden pheasant'
}

bird; // 'mandarin duck'
```

*bird* is scoped to  
*birdWatch* function

# BLOCK SCOPE

```
let radius = 8;

if(radius > 0){

    const PI = 3.14;

    let circ = 2 * PI * radius;

}

console.log(radius); //8
console.log(PI); //NOT DEFINED
console.log(circ); //NOT DEFINED
```

*PI* & *circ* are  
scoped to the  
BLOCK

# LEXICAL SCOPE

```
● ● ●  
function outer() {  
  let hero = "Black Panther";  
  
  function inner() {  
    let cryForHelp = `${hero}, please save me!`  
    console.log(cryForHelp);  
  }  
  
  inner();  
}
```

# FUNCTION EXPRESSIONS

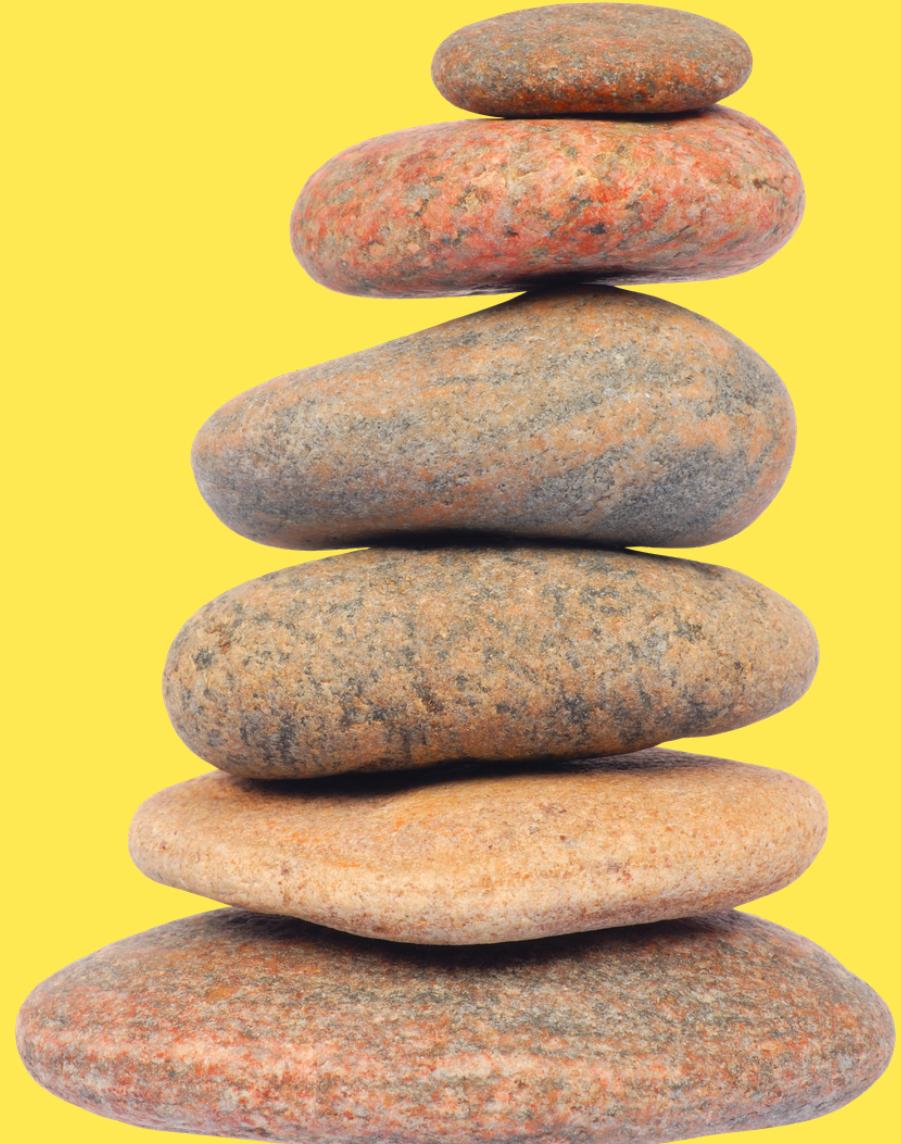


```
const square = function (num) {  
    return num * num;  
}  
square(7); //49
```

**FUNCTIONS  
ARE...  
OBJECTS!**



# HIGHER ORDER FUNCTIONS



# HIGHER ORDER FUNCTIONS

Functions that operate on/with other functions.

They can:

- Accept other functions as arguments
- Return a function

# FUNCTIONS AS ARGUMENTS

```
● ● ●  
  
function callTwice(func) {  
  func();  
  func();  
}  
  
function laugh() {  
  console.log("HAHAHAHAHAHAHAHAHAHAHAH");  
}  
callTwice(laugh) //pass a function as an arg!  
// "HAHAHAHAHAHAHAHHHAHAHAH"  
// "HAHAHAHAHAHAHAHHHAHAHAH"
```

# RETURNING FUNCTIONS

```
function makeBetweenFunc(min, max) {  
  return function (val) {  
    return val >= min && val <= max;  
  }  
}  
  
const inAgeRange = makeBetweenFunc(18, 100);  
  
inAgeRange(17); //false  
inAgeRange(68); //true
```

# METHODS



```
const math = {  
    multiply : function(x, y) {  
        return x * y;  
    },  
    divide   : function(x, y) {  
        return x / y;  
    },  
    square   : function(x) {  
        return x * x;  
    }  
};
```

We can add functions as properties on objects.

We call them **methods!**

# SHORTHAND

```
const math = {  
    blah: 'Hi!',  
    add(x, y) {  
        return x + y;  
    },  
    multiply(x, y) {  
        return x * y;  
    }  
}  
math.add(50, 60) //110
```

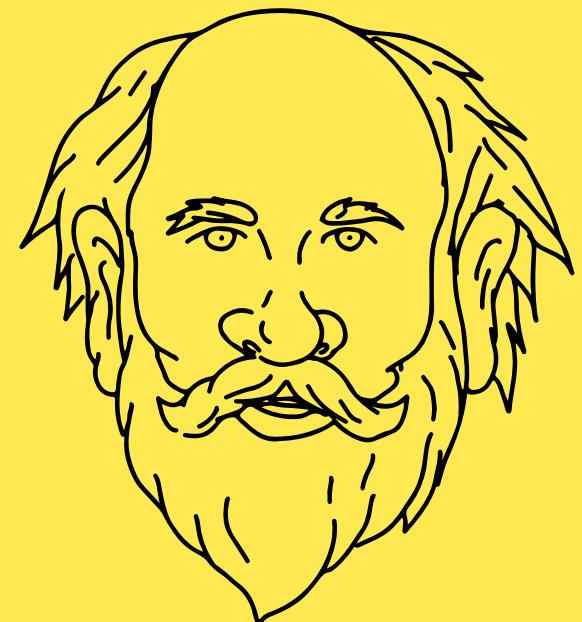
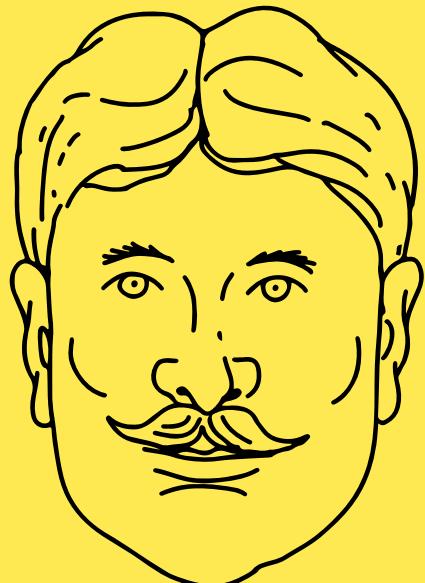
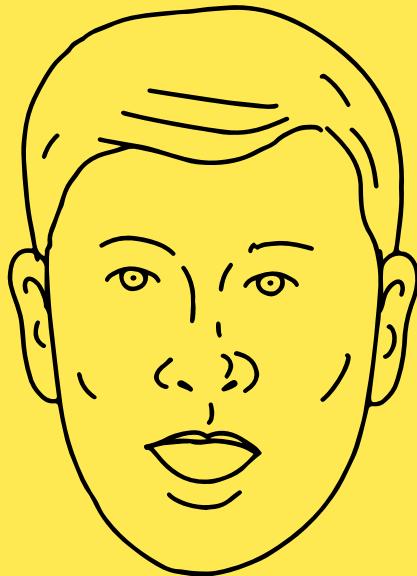
We do this so often that there's a new shorthand way of adding methods.

# 'THIS' IN METHODS

Use the keyword `this` to access other properties on the same object.

```
const person = {  
    first: 'Robert',  
    last: 'Herjavec',  
    fullName() {  
        return `${this.first} ${this.last}`  
    }  
}  
person.fullName(); // "Robert Herjavec"  
person.last = "Plant";  
person.fullName(); // "Robert Plant"
```

The value of ***this*** depends on  
the invocation context of  
the function it is used in.



# SAME FUNCTION



```
const person = {  
  first: 'Robert',  
  last: 'Herjavec',  
  fullName() {  
    return `${this.first} ${this.last}`  
  }  
}
```



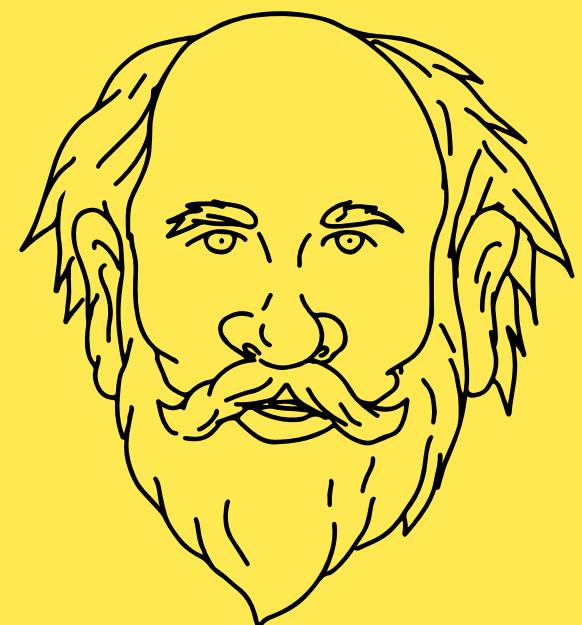
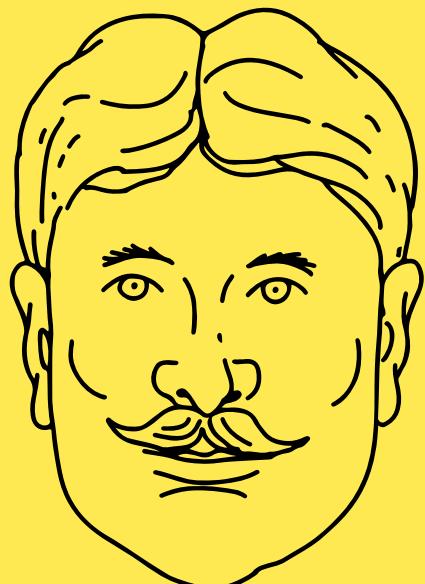
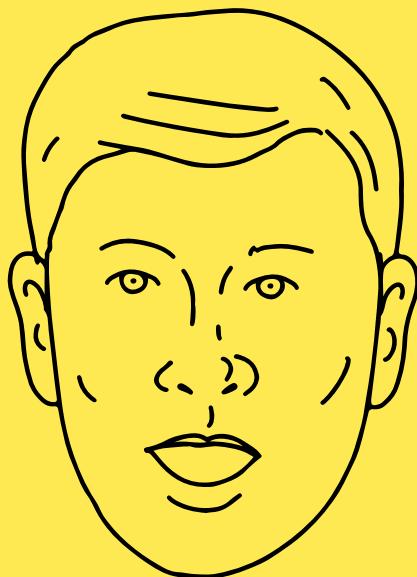
```
person.fullName();  
// "Robert Herjavec"
```

# DIFFERENT RESULT? ? ?



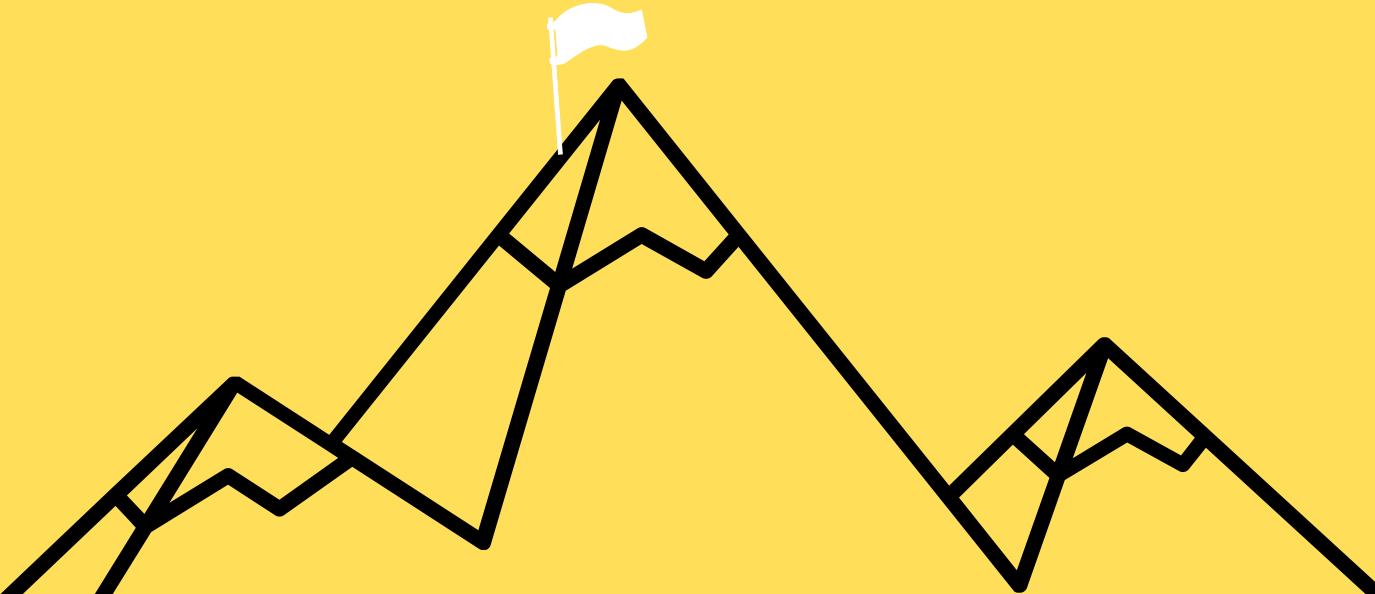
```
const func = person.fullName;  
func()  
// "undefined undefined"
```

The value of ***this*** depends on  
the **invocation context** of  
the function it is used in.



# GOALS

- Use the new arrow function syntax
- Understand and use these methods:
  - forEach
  - map
  - filter
  - find
  - reduce
  - some
  - every



# FOREACH



```
const nums = [9, 8, 7, 6, 5, 4, 3, 2, 1];

nums.forEach(function (n) {
  console.log(n * n)
  //prints: 81, 64, 49, 36, 25, 16, 9, 4, 1
});

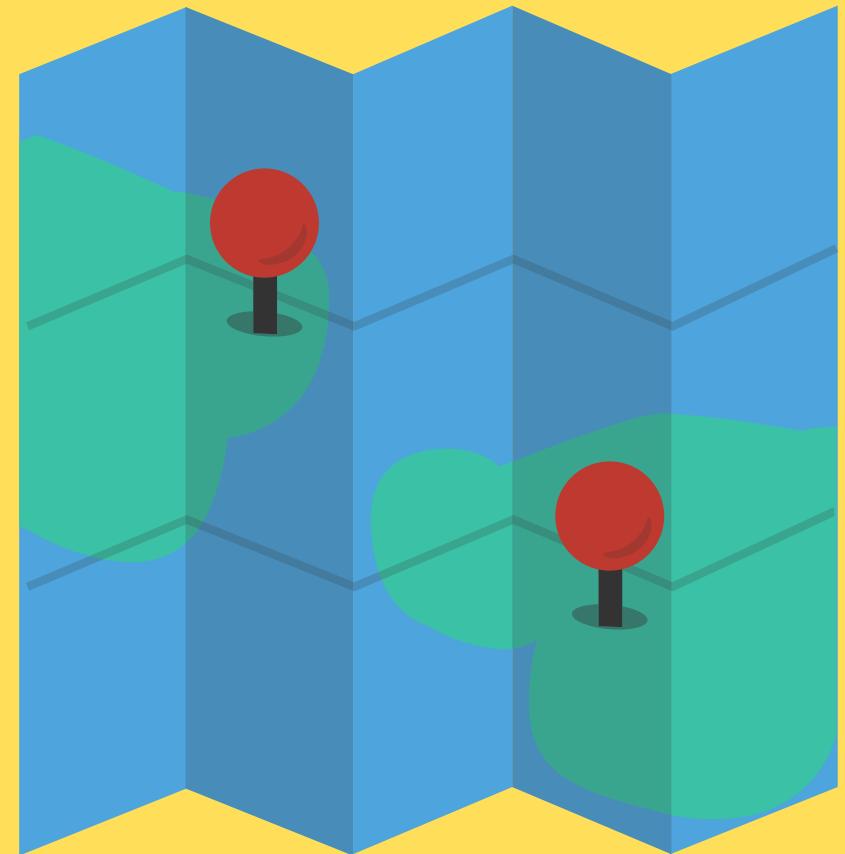
nums.forEach(function (el) {
  if (el % 2 === 0) {
    console.log(el)
    //prints: 8, 6, 4, 2
  }
})
```

Accepts a callback function.

Calls the function once per element in the array.

# MAP

Creates a new array with the results of calling a callback on every element in the array

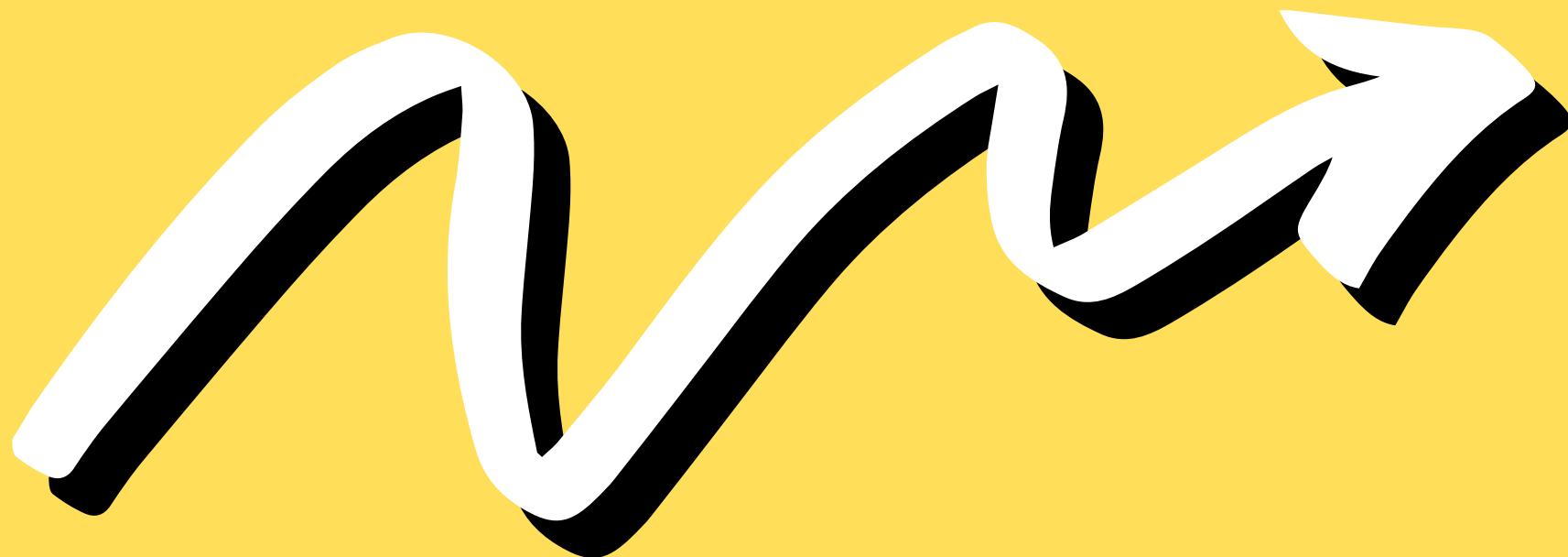


# MAP



```
const texts = ['rofl', 'lol', 'omg', 'ttyl'];
const caps = texts.map(function (t) {
  return t.toUpperCase();
})
texts; //["rofl", "lol", "omg", "ttyl"]
caps; //["ROFL", "LOL", "OMG", "TTYL"]
```

# ARROW FUNCTIONS!



# ARROW FUNCTIONS

"syntactically compact alternative"  
to a regular function expression



```
const square = (x) => {
  return x * x;
}
```

```
const sum = (x, y) => {
  return x + y;
}
```

# ARROW FUNCTIONS



```
//parens are optional if there's only one parameter:  
const square = x => {  
    return x * x;  
}  
  
//Use empty parens for functions w/ no parameters:  
const singASong = () => {  
    return "LA LA LA LA LA LA";  
}
```

# IMPLICIT RETURN

All these functions do the same thing:

```
● ● ●

const isEven = function (num) { //regular function expression
  return num % 2 === 0;
}
const isEven = (num) => { //arrow function with parens around param
  return num % 2 === 0;
}
const isEven = num => { //no parens around param
  return num % 2 === 0;
}
const isEven = num => ( //implicit return
  num % 2 === 0
);
const isEven = num => num % 2 === 0; //one-liner implicit return
```

# FIND

returns the value of the first element in the array that satisfies the provided testing function.

```
let movies = [
  "The Fantastic Mr. Fox",
  "Mr. and Mrs. Smith",
  "Mrs. Doubtfire",
  "Mr. Deeds"
]
let movie = movies.find(movie => {
  return movie.includes('Mrs.')
}) //"Mr. and Mrs. Smith"

let movie2 = movies.find(m => m.indexOf('Mrs') === 0);
// "Mrs. Doubtfire"
```

# FILTER

Creates a new array with all elements that pass the test implemented by the provided function.

```
● ● ●  
  
const nums = [9, 8, 7, 6, 5, 4, 3, 2, 1];  
const odds = nums.filter(n => {  
    return n % 2 === 1; //our callback returns true or false  
    //if it returns true, n is added to the filtered array  
})  
//[9, 7, 5, 3, 1]  
  
const smallNums = nums.filter(n => n < 5);  
//[4, 3, 2, 1]
```

# EVERY

tests whether **all** elements in the array pass the provided function. It returns a Boolean value.



```
const words = ["dog", 'dig', 'log', 'bag', 'wag'];

words.every(word => {
  return word.length === 3;
}) //true

words.every(word => word[0] === 'd'); //false

words.every(w => {
  let last_letter = w[w.length - 1];
  return last_letter === 'g'
}) //true
```

# SOME

Similar to every, but returns true if ANY of the array elements pass the test function

```
● ● ●

const words = ['dog', 'jello', 'log', 'cupcake', 'bag', 'wag'];

//Are there any words longer than 4 characters?
words.some(word => {
  return word.length > 4;
}) //true

//Do any words start with 'Z'?
words.some(word => word[0] === 'Z'); //false

//Do any words contain 'cake'?
words.some(w => w.includes('cake')) //true
```

# REDUCE

Executes a reducer function on each element of the array, resulting in a single value.



# SUMMING AN ARRAY



```
[3, 5, 7, 9, 11].reduce((accumulator, currentValue) => {  
    return accumulator + currentValue;  
});
```

Callback	accumulator	currentValue	return value
first call	3	5	8
second call	8	7	15
third call	15	9	24
fourth call	24	11	35

# FINDING MAX VAL

```
let grades = [89, 96, 58, 77, 62, 93, 81, 99, 73];

const topScore = grades.reduce((max, currVal) => {
  if (currVal > max) return currVal;
  return max;
})
topScore; //99

//A shorter option w/ Math.max & implicit return
const topScore = grades.reduce((max, currVal) => (
  Math.max(max, currVal)
))
```

# INITIAL VALUE



```
[4, 5, 6, 7, 8].reduce((accumulator, currentValue) => {  
    return accumulator + currentValue;  
});  
//RETURNS: 30
```

```
[4, 5, 6, 7, 8].reduce((accumulator, currentValue) => {  
    return accumulator + currentValue;  
}, 100);  
//RETURNS: 130
```

# TALLYING



```
const votes =  
['y', 'y', 'n', 'y', 'n', 'y', 'n', 'y', 'n', 'n', 'y', 'y'];  
const tally = votes.reduce((tally, vote) => {  
  tally[vote] = (tally[vote] || 0) + 1;  
  return tally;  
}, {}); //INITIAL VALUE: {}  
  
tally; //{y: 7, n: 6}
```

# DEFAULT PARAMS

## The Old Way

```
function multiply(a, b) {  
  b = typeof b !== 'undefined' ? b : 1;  
  return a * b;  
}  
  
multiply(7); //7  
multiply(7, 3); //21
```

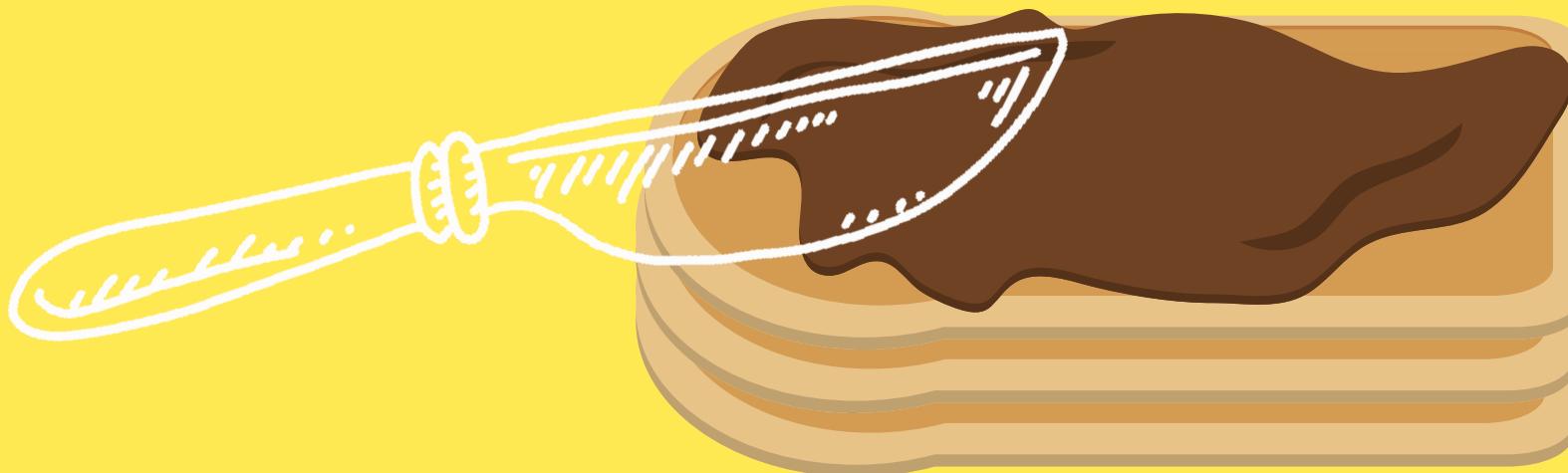
# DEFAULT PARAMS

The New Way



```
function multiply(a, b = 1) {  
    return a * b;  
}  
  
multiply(4); //4  
multiply(4, 5); //20
```

# SPREAD



# SPREAD

Spread syntax allows an iterable such as an array to be **expanded** in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected.

WHAT?

# SPREAD

## For Function Calls



```
const nums = [ 9, 3, 2, 8 ];  
Math.max(nums); //NaN  
// Use spread!  
Math.max(...nums); //67  
// Same as calling:  
// Math.max(9,3,2,8)
```

Expands an iterable  
(array, string, etc.)  
into a list of arguments



```
const nums1 = [ 1, 2, 3 ];  
const nums2 = [ 4, 5, 6 ];  
  
[ ...nums1, ...nums2 ];  
//[1, 2, 3, 4, 5, 6]  
  
[ 'a', 'b', ...nums2 ];  
//[ "a", "b", 4, 5, 6]  
  
[ ...nums1, ...nums2, 7, 8, 9 ];  
//[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

# SPREAD

## In Array Literals

Create a new array using an existing array. Spreads the elements from one array into a new array.

# SPREAD

## In Object Literals

```
● ● ●  
const feline = { legs: 4, family: 'Felidae' };  
const canine = { family: 'Caninae', furry: true };  
  
const dog = { ...canine, isPet: true };  
//{family: "Caninae", furry: true, isPet: true}  
  
const lion = { ...feline, genus: 'Panthera' };  
//{legs: 4, family: "Felidae", genus: "Panthera"}  
  
const catDog = { ...feline, ...canine };  
//{legs: 4, family: "Caninae", furry: true}
```

Copies properties  
from one object into  
another object literal.

# REST

It looks like spread,  
but it's not!



# THE ARGUMENTS OBJECT

```
function sumAll() {  
    let total = 0;  
    for (let i = 0; i < arguments.length; i++)  
    {        total += arguments[i];  
    }  
    return total;  
}  
sumAll(8, 4, 3, 2); // 17  
sumAll(2, 3); //5
```

- Available inside every function.
- It's an **array-like** object
  - Has a `length` property
  - Does not have array methods like `push/pop`
- Contains all the arguments passed to the function
- Not available inside of arrow functions!

# REST PARAMS

Collects all remaining arguments into an actual array



```
function sumAll(...nums) {  
  let total = 0;  
  for (let n of nums) total += n;  
  return total;  
  
sumAll(1, 2); //3  
sumAll(1, 2, 3, 4, 5); //15
```

# DESTRUCTURING

A short, clean syntax to 'unpack':

- Values from arrays
- Properties from objects

Into distinct variables.



# ARRAY

## Destructuring



```
const raceResults = [ 'Eliud Kipchoge', 'Feyisa Lelisa', 'Galen Rupp' ];

const [ gold, silver, bronze ] = raceResults;
gold; // "Eliud Kipchoge"
silver; // "Feyisa Lelisa"
bronze; // "Galen Rupp"

const [ fastest, ...everyoneElse ] = raceResults;
fastest; // "Eliud Kipchoge"
everyoneElse; // ["Feyisa Lelisa", "Galen Rupp"]
```

# OBJECT

## Destructuring



```
const runner = {  
    first: "Eliud",  
    last: "Kipchoge",  
    country: "Kenya",  
    title: "Elder of the Order of the Golden Heart of Kenya"  
}  
const {first, last, country} = runner;  
  
first; // "Eliud"  
last; // "Kipchoge"  
country; // "Kenya"
```

# PARAM

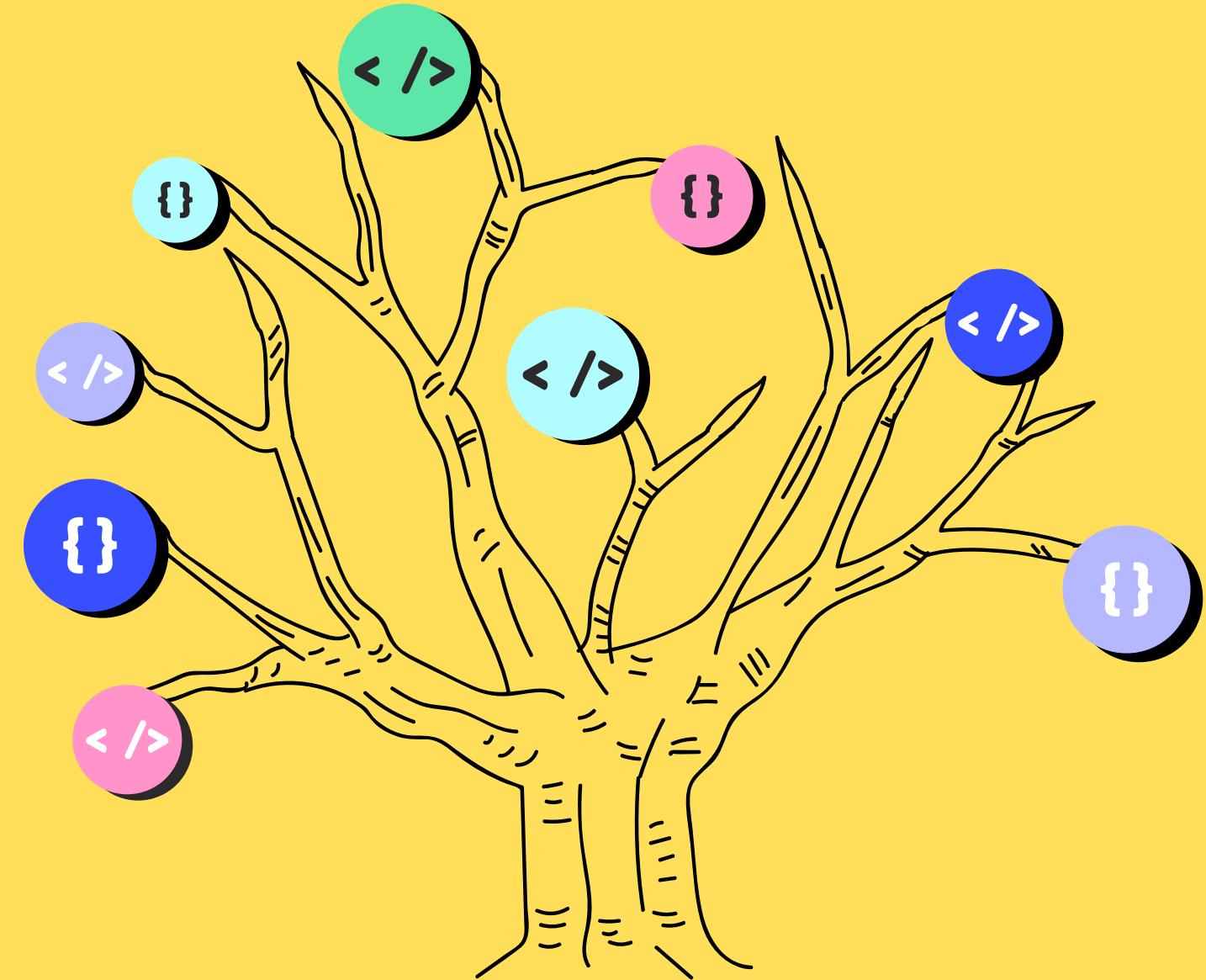
## Destructuring

```
● ● ●

const fullName = ({first, last}) => {
  return `${first} ${last}`
}
const runner = {
  first: "Eliud",
  last: "Kipchoge",
  country: "Kenya",
}

fullName(runner); // "Eliud Kipchoge"
```

# THE DOM



# DOCUMENT

# OBJECT

# MODEL



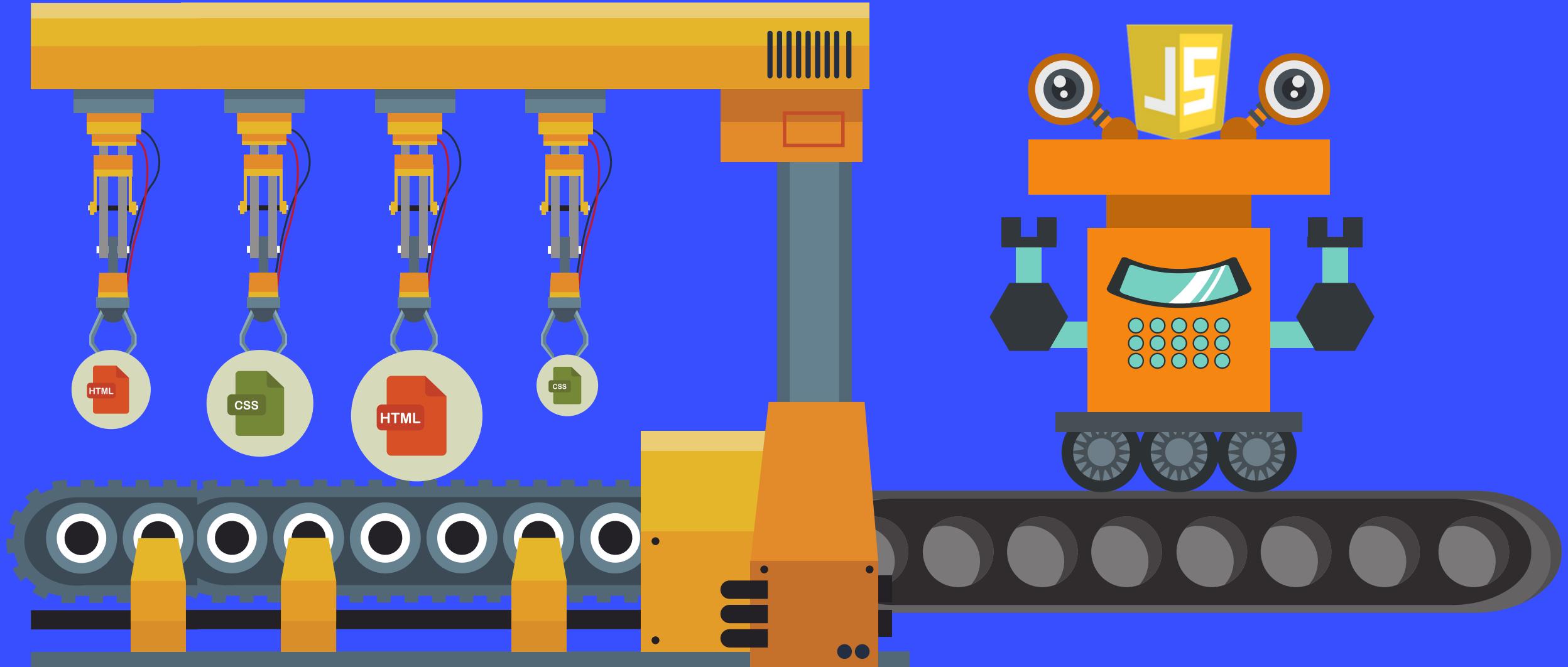
# WHAT IS IT?

- The DOM is a JavaScript representation of a webpage.
- It's your JS "window" into the contents of a webpage
- It's just a bunch of objects that you can interact with via JS.



HTML+CSS Go In...

JS Objects Come Out





```
<body>
  <h1>Hello!</h1>
  <ul>
    <li>Water Plants</li>
    <li>Get Some Sleep</li>
  </ul>
</body>
```



I'm an object!

DOCUMENT

BODY

H1

UL

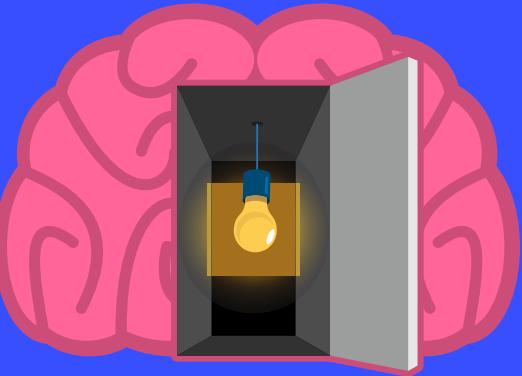
LI

LI

Me too!

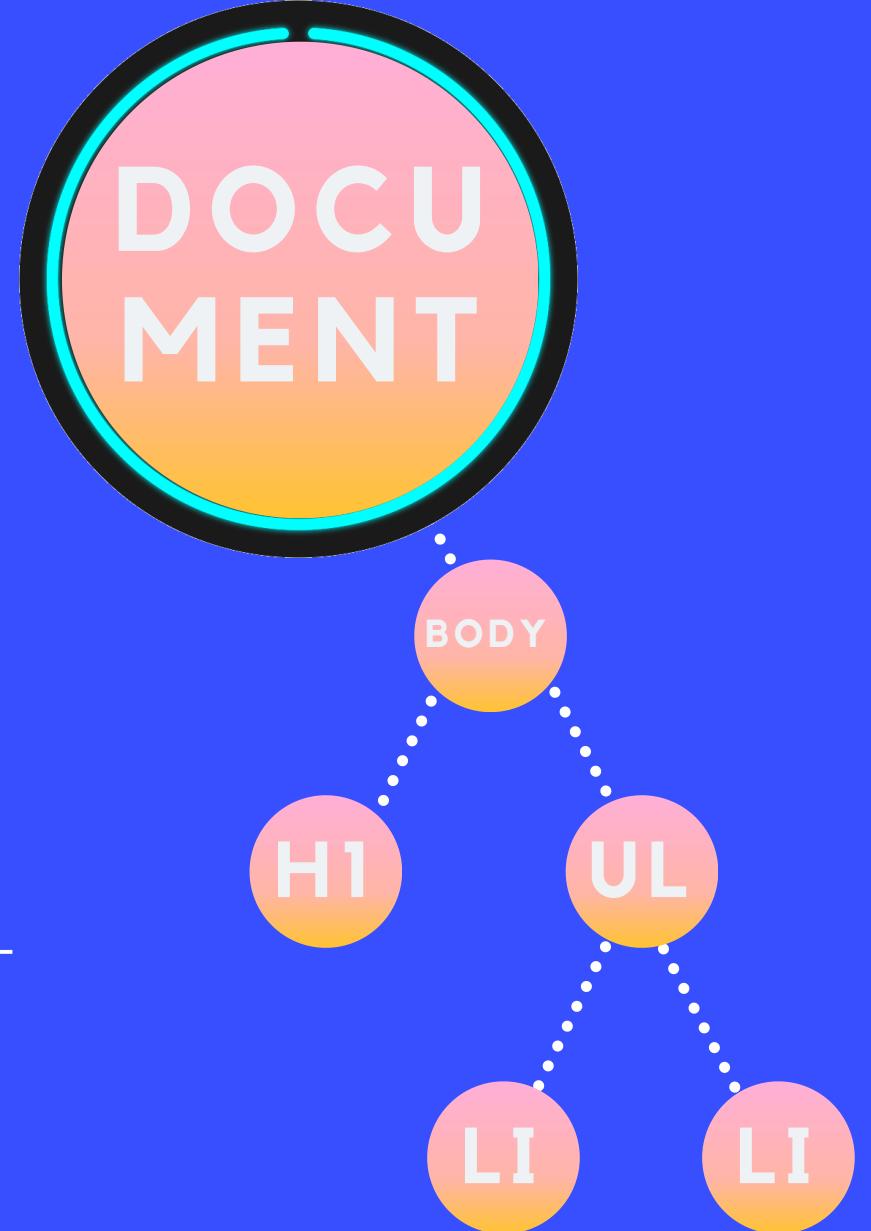
HTML+CSS Go In...

JS Objects Come Out



# DOCUMENT

The document object is our entry point into the world of the DOM. It contains representations of all the content on a page, plus tons of useful methods and properties



# SELECTING



**1**

**SELECT**

**2**

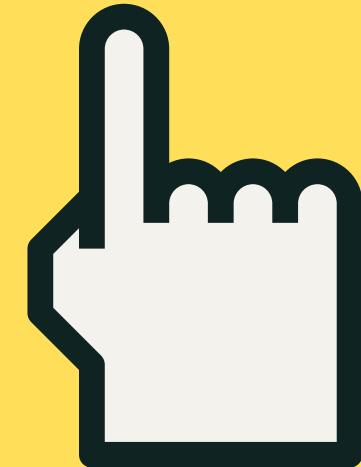
**MANIPULATE**



# SELECTING



- getElementById
- getElementsByTagName
- getElementsByClassName



# querySelector

- A newer, all-in-one method to select a single element.

```
...  
//Finds first h1 element:  
document.querySelector('h1');  
  
//Finds first element with ID of red:  
document.querySelector('#red');  
  
//Finds first element with class of  
document.querySelector('.big');
```



# querySelectorAll

Same idea , but returns a collection of matching elements

**1**

SELECT

**2**

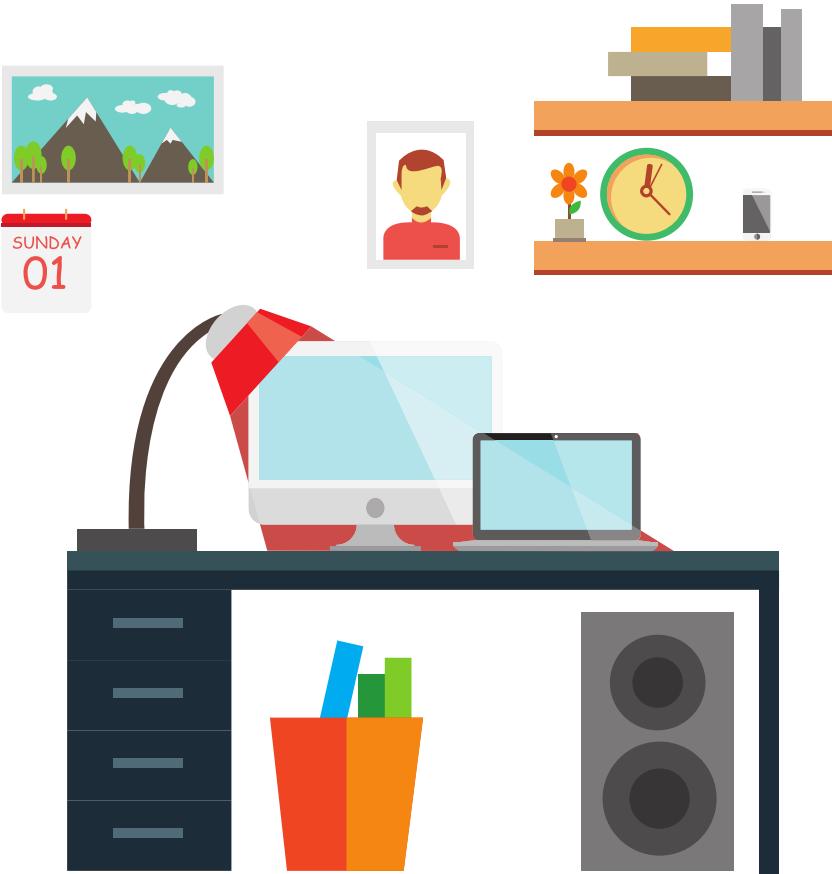
MANIPULATE



# PROPERTIES & METHODS

(the important ones)

- classList
- getAttribute()
- setAttribute()
- appendChild()
- append()
- prepend()
- removeChild()
- remove()
- createElement



- innerText
- .textContent
- innerHTML
- value
- parentElement
- children
- nextSibling
- previousSibling
- style

# EVENTS

Responding to  
user inputs  
and actions!



# A SMALL TASTE

- clicks
- drags
- drops
- hovers
- scrolls
- form submission
- key presses
- focus/blur



- mouse wheel
- double click
- copying
- pasting
- audio start
- screen resize
- printing

# addEventListener

Specify the event type and a callback to run



```
const button = document.querySelector('h1');

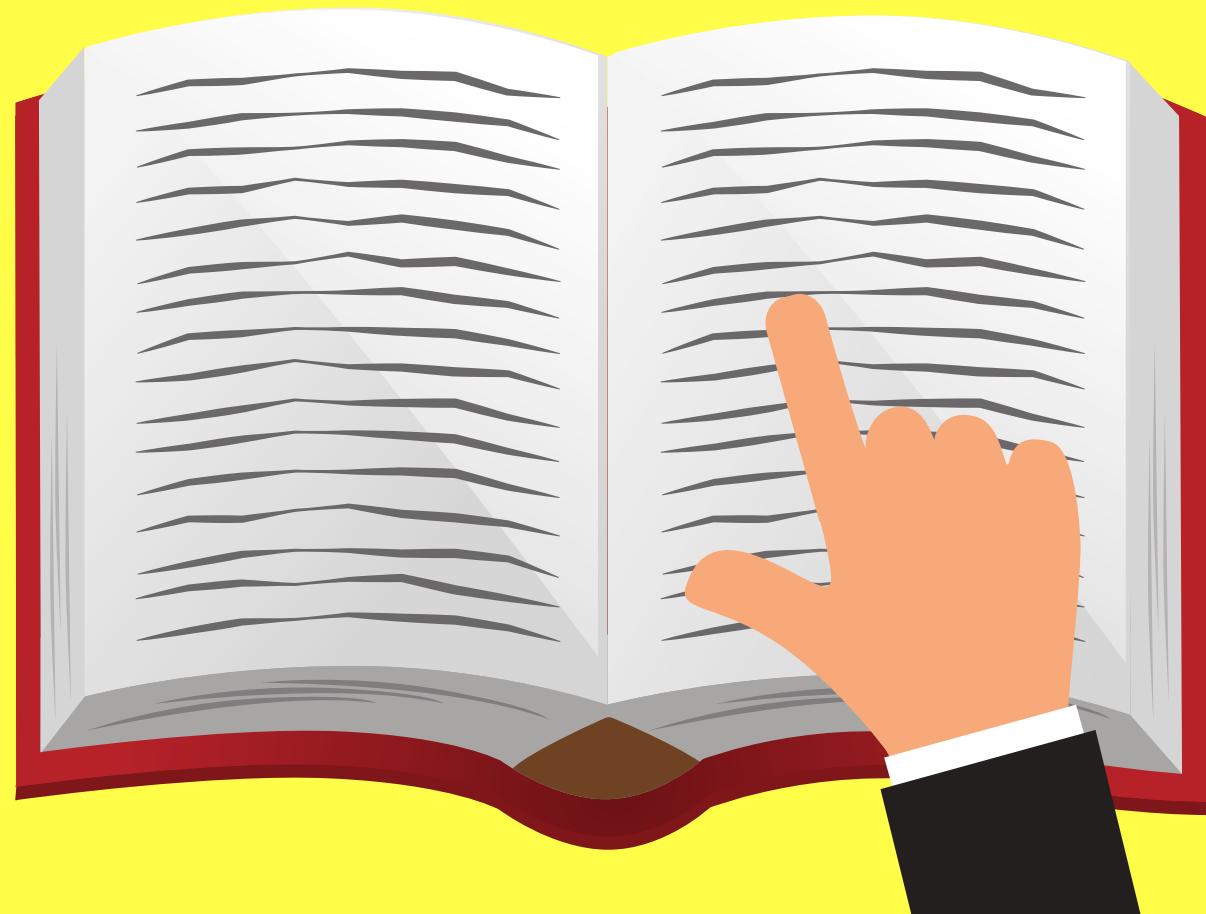
button.addEventListener('click', () => {
  alert("You clicked me!!")
})
```

# CALL STACK

The mechanism the JS interpreter uses to keep track of its place in a script that calls multiple functions.

How JS "knows" what function is currently being run and what functions are called from within that function, etc.

# CALL STACK



Let's see...  
where was I?!

LAST  
THING  
IN...



FIRST  
THING  
OUT...



# HOW IT WORKS

- When a script calls a function, the interpreter adds it to the call stack and then starts carrying out the function.
- Any functions that are called by that function are added to the call stack further up, and run where their calls are reached.
- When the current function is finished, the interpreter takes it off the stack and resumes execution where it left off in the last code listing.



```
const multiply = (x, y) => x * y;  
  
const square = (x) => multiply(x, x);  
  
→ const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
  
isRightTriangle(3, 4, 5);
```

isRightTriangle(3,4,5)  
square(3)+square(4)  
==== square(5)



```
const multiply = (x, y) => x * y;  
→const square = (x) => multiply(x, x);  
const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
isRightTriangle(3, 4, 5);
```

**square(3)**  
multiply(3,3)

**isRightTriangle(3,4,5)**  
square(3)+square(4)  
== square(5)

• • •

```
→ const multiply = (x, y) => x * y;  
  
const square = (x) => multiply(x, x);  
  
const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
  
isRightTriangle(3, 4, 5);
```

**multiply(3,3)**

9

**square(3)**

**multiply(3,3)**

**isRightTriangle(3,4,5)**  
**square(3)+square(4)**  
**== square(5)**



```
const multiply = (x, y) => x * y;  
→const square = (x) => multiply(x, x);  
const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
isRightTriangle(3, 4, 5);
```

**square(3)**

9

**isRightTriangle(3,4,5)**  
**square(3)+square(4)**  
**== square(5)**



```
const multiply = (x, y) => x * y;  
  
const square = (x) => multiply(x, x);  
  
→ const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
  
isRightTriangle(3, 4, 5);
```

isRightTriangle(3,4,5)  
9+square(4) === square(5)



```
const multiply = (x, y) => x * y;  
→ const square = (x) => multiply(x, x);  
const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
isRightTriangle(3, 4, 5);
```

**square(4)**

**multiply(4,4)**

**isRightTriangle(3,4,5)**

**9+square(4) === square(5)**



```
const multiply = (x, y) => x * y;  
  
const square = (x) => multiply(x, x);  
  
const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
  
isRightTriangle(3, 4, 5);
```

**multiply(4,4)**  
16

**square(4)**  
multiply(4,4)

**isRightTriangle(3,4,5)**  
9+square(4) === square(5)



```
const multiply = (x, y) => x * y;  
→ const square = (x) => multiply(x, x);  
const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
isRightTriangle(3, 4, 5);
```

**square(4)**

16

**isRightTriangle(3,4,5)**  
**9+square(4) === square(5)**



```
const multiply = (x, y) => x * y;  
  
const square = (x) => multiply(x, x);  
  
→ const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
  
isRightTriangle(3, 4, 5);
```

isRightTriangle(3,4,5)  
9+16 === square(5)



```
const multiply = (x, y) => x * y;  
→ const square = (x) => multiply(x, x);  
const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
isRightTriangle(3, 4, 5);
```

**square(5)**  
multiply(5,5)

**isRightTriangle(3,4,5)**  
9+16 === **square(5)**



```
const multiply = (x, y) => x * y;  
  
const square = (x) => multiply(x, x);  
  
const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
  
isRightTriangle(3, 4, 5);
```

multiply(5,5)  
25

square(5)  
multiply(5,5)

isRightTriangle(3,4,5)  
9+16 === square(5)



```
const multiply = (x, y) => x * y;  
→const square = (x) => multiply(x, x);  
const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
isRightTriangle(3, 4, 5);
```

**square(5)**

**25**

**isRightTriangle(3,4,5)**  
 $9+16 === \text{square}(5)$



```
const multiply = (x, y) => x * y;  
  
const square = (x) => multiply(x, x);  
  
→ const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
  
isRightTriangle(3, 4, 5);
```

isRightTriangle(3,4,5)

9+16 === 25



```
const multiply = (x, y) => x * y;  
  
const square = (x) => multiply(x, x);  
  
→ const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
  
isRightTriangle(3, 4, 5);
```

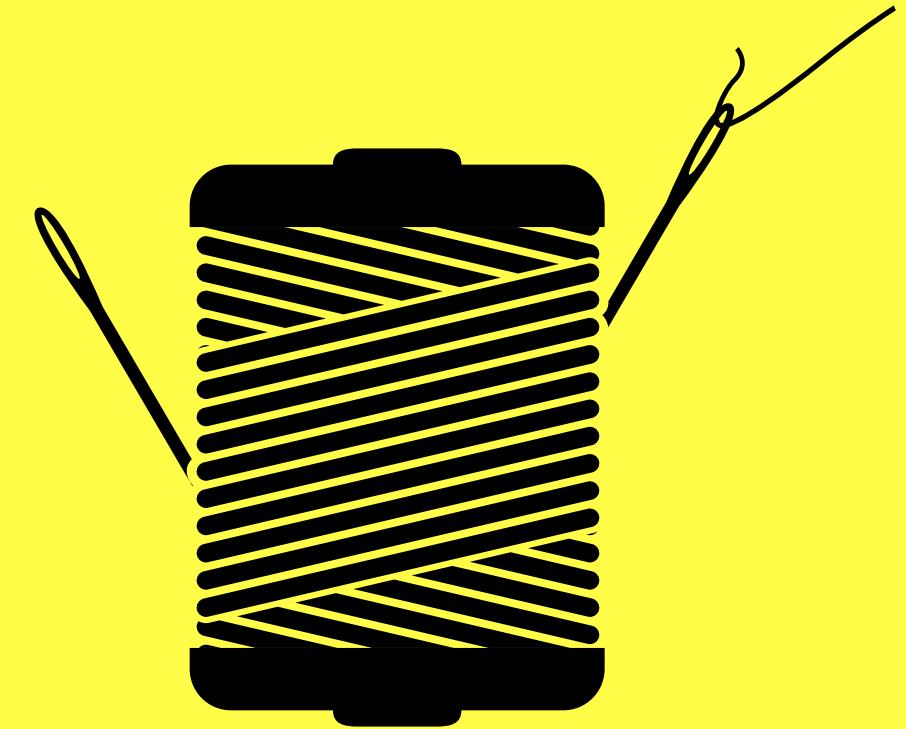
isRightTriangle(3,4,5)  
true



```
const multiply = (x, y) => x * y;  
  
const square = (x) => multiply(x, x);  
  
→ const isRightTriangle = (a, b, c) => {  
    return square(a) + square(b) === square(c);  
};  
  
isRightTriangle(3, 4, 5);
```

true

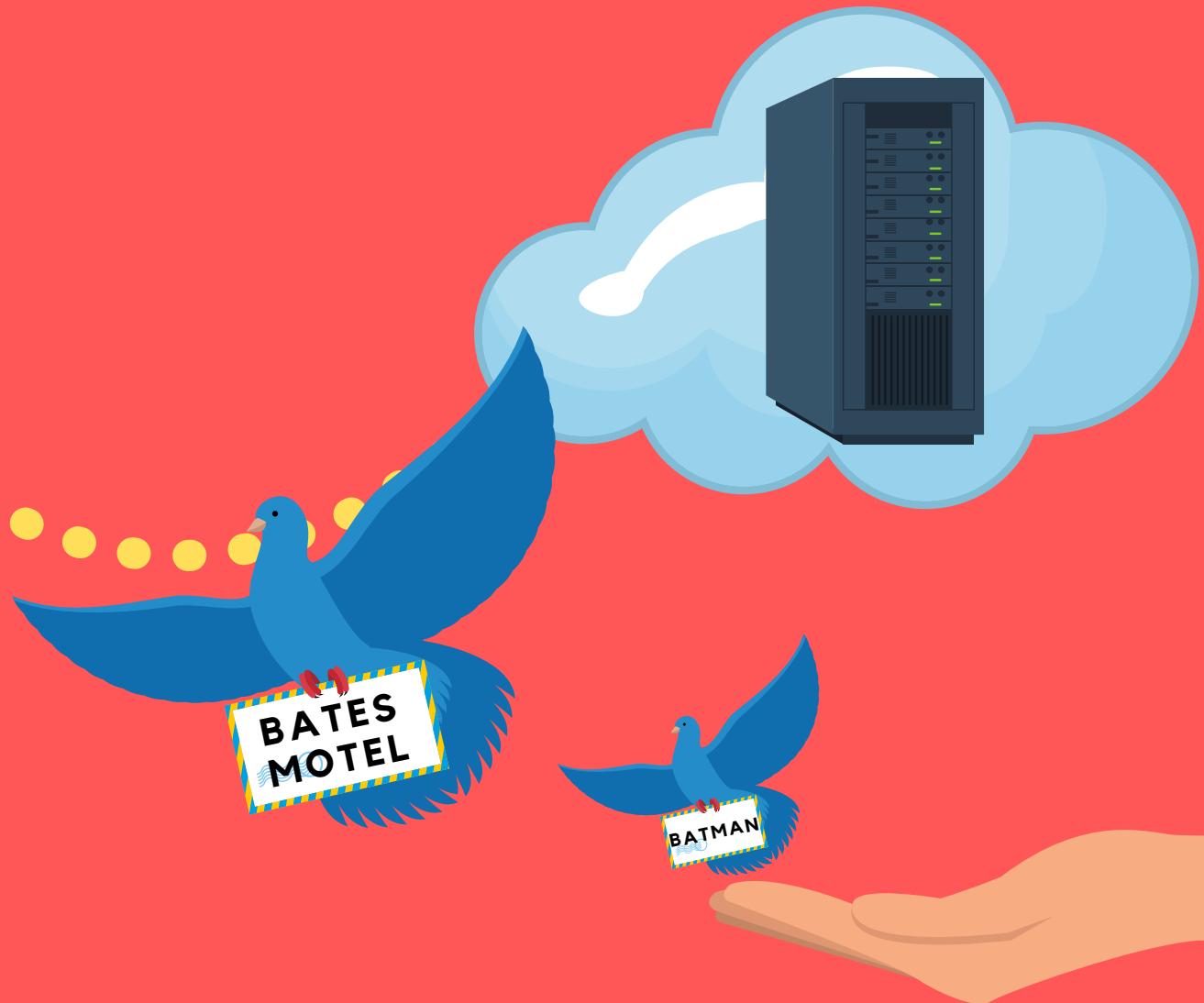
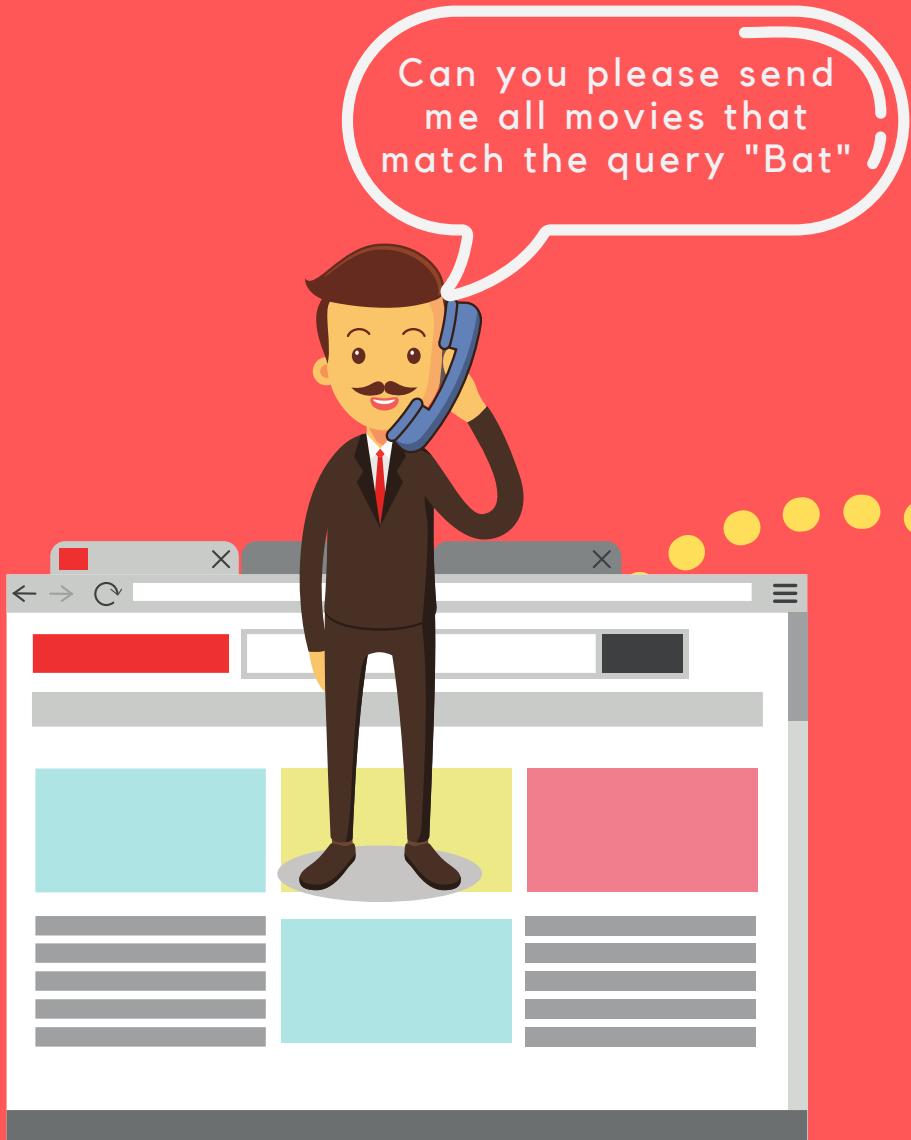
JS IS  
SINGLE  
THREADED

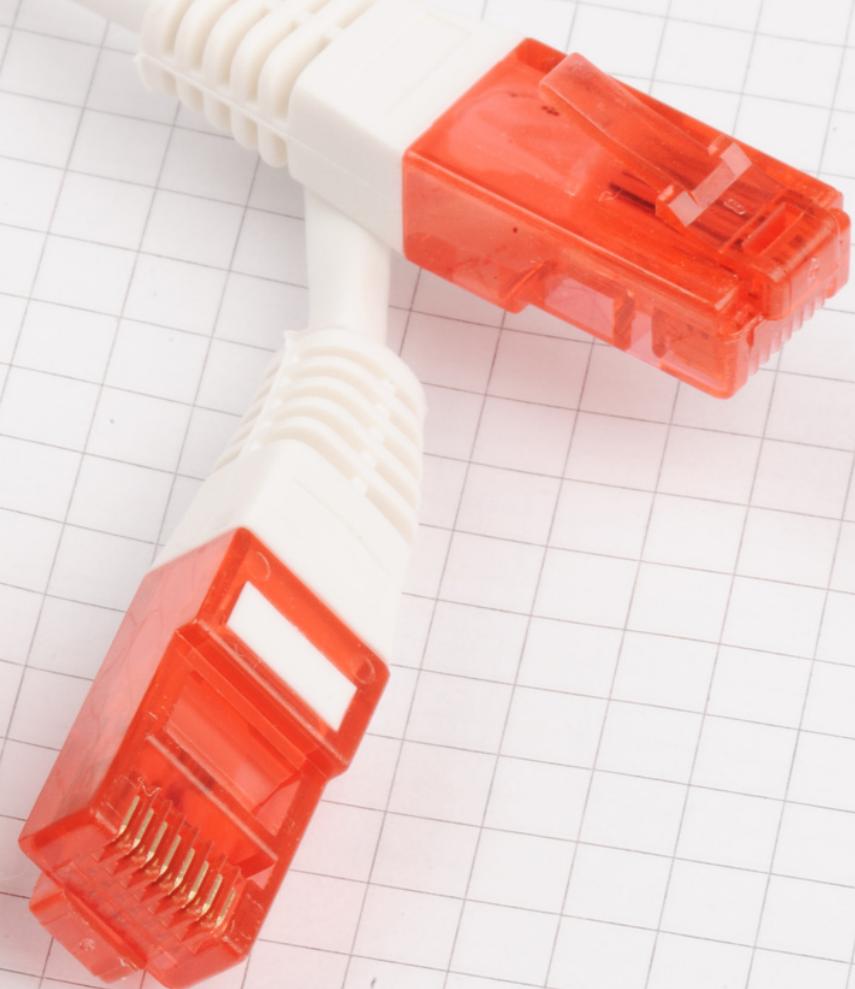


# **WHAT DOES THAT MEAN?**

At any given point in time, that single JS thread is running at most one line of JS code.







**THIS TAKES TIME**

**IS OUR APP  
GOING TO  
GRIND TO  
A HALT?**



# What happens when something takes a long time?



```
const newTodo = input.value; //get user input  
saveToDatabase(newTodo); //this could take a while!  
input.value = ''; //reset form
```

Fortunately...  
We have a workaround

```
● ● ●  
console.log('I print first!');  
setTimeout(() => {  
  console.log('I print after 3 seconds');  
, 3000);  
console.log('I print second!');
```



CALLBACKS????!

THE  
BROWSER  
DOES THE  
WORK!



# OK BUT HOW?

- Browsers come with Web APIs that are able to handle certain tasks in the background (like making requests or setTimeout)
- The JS call stack recognizes these Web API functions and passes them off to the browser to take care of
- Once the browser finishes those tasks, they return and are pushed onto the stack as a callback.

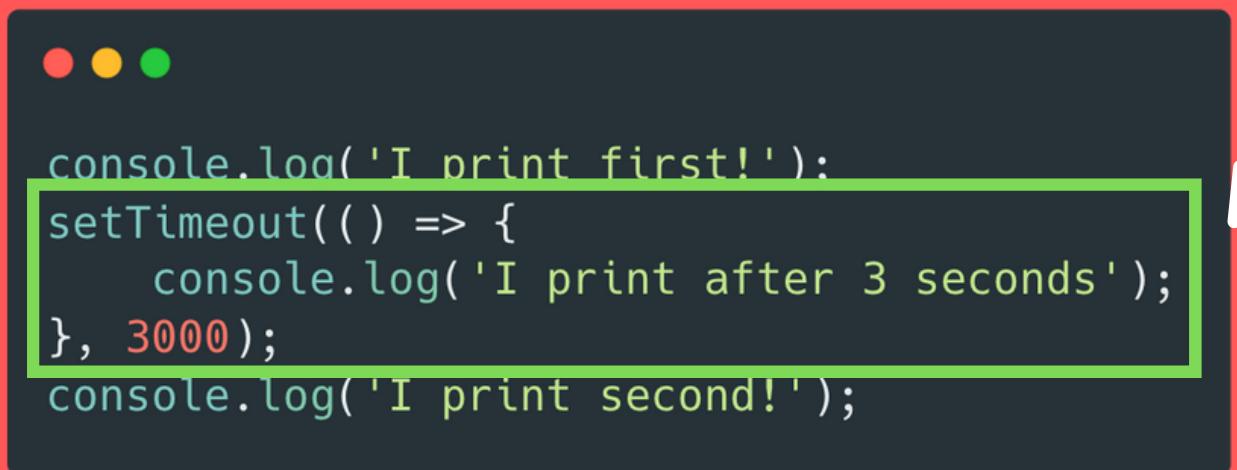


> I print first!

```
● ● ●  
console.log('I print first!');  
setTimeout(() => {  
    console.log('I print after 3 seconds');  
}, 3000);  
console.log('I print second!');
```



```
> I print first!
```



```
console.log('I print first!');  
setTimeout(() => {  
    console.log('I print after 3 seconds');  
}, 3000);  
console.log('I print second!');
```



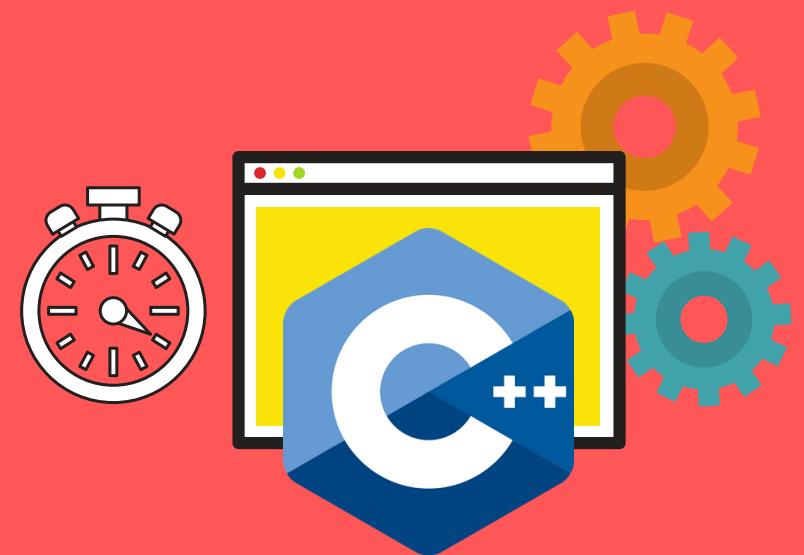
Hey browser, can you  
set a timer for 3  
seconds?

OKEEEDOKEEE



> I print first!  
> I print second!

```
● ● ●  
console.log('I print first!');  
setTimeout(() => {  
    console.log('I print after 3 seconds');  
}, 3000);  
console.log('I print second!');
```



- > I print first!
- > I print second!

```
● ● ●  
console.log('I print first!');  
setTimeout(() => {  
    console.log('I print after 3 seconds');  
, 3000);  
console.log('I print second!');
```



Will do!  
Thanks, browser!

Time's Up!!!  
Make sure you run  
that callback now!!



- > I print first!
- > I print second!
- > I print after 3 seconds!

```
● ● ●  
console.log('I print first!')  
setTimeout(() => {  
    console.log('I print after 3 seconds');  
, 3000);  
console.log('I print second!');
```



```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename,
function(err) {
  if (err) console.log('Error writing file: ' + err)
  })
  .bind(this))
}
})
})
})
})
```

# Callback Hell



# ENTER PROMISES

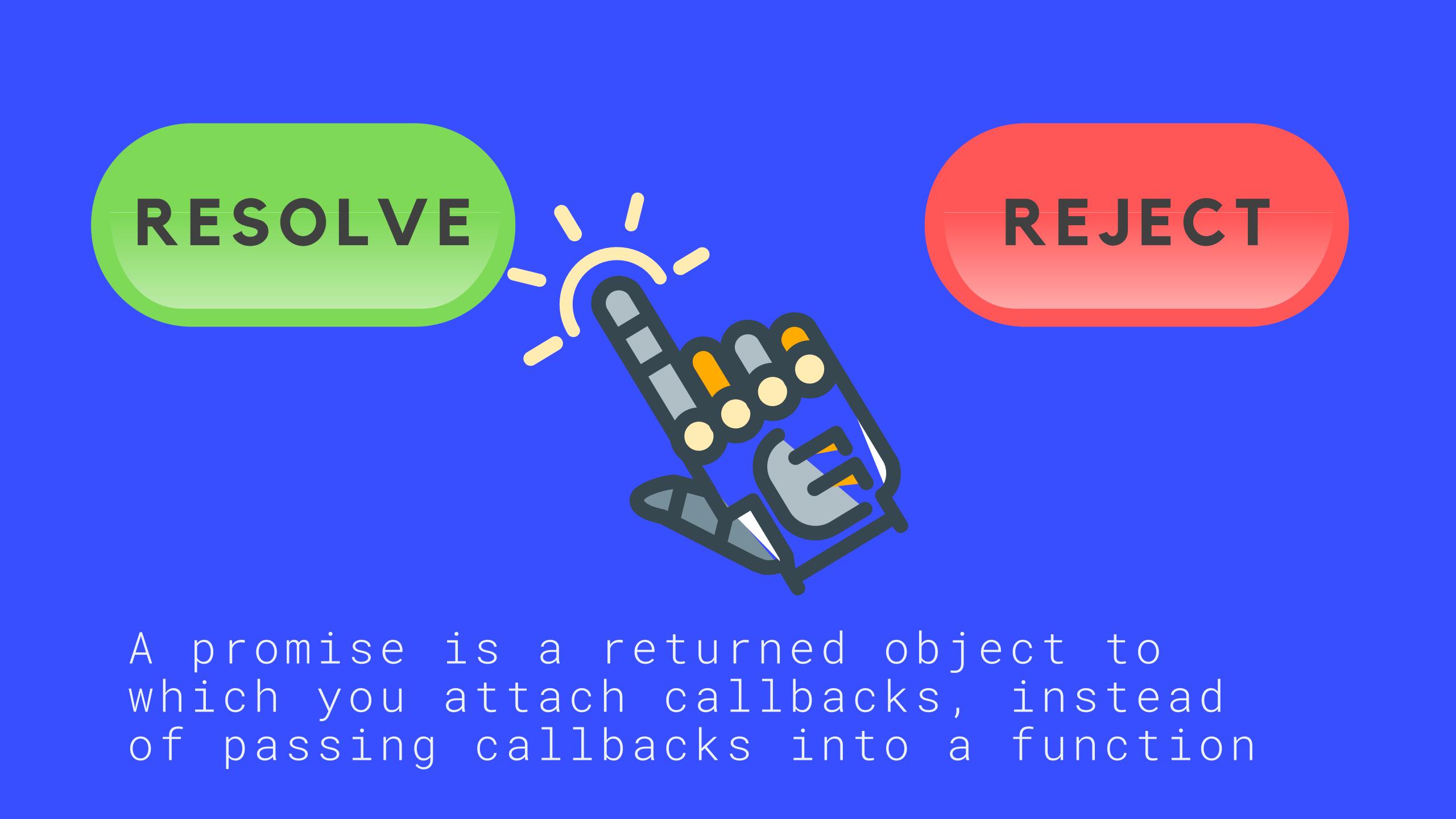
A Promise is an object representing the eventual completion or failure of an asynchronous operation



# PROMISES

A pattern  
for writing  
async code.





# RESOLVE

# REJECT

A promise is a returned object to which you attach callbacks, instead of passing callbacks into a function

loadRedditPosts (not shown)  
returns a promise



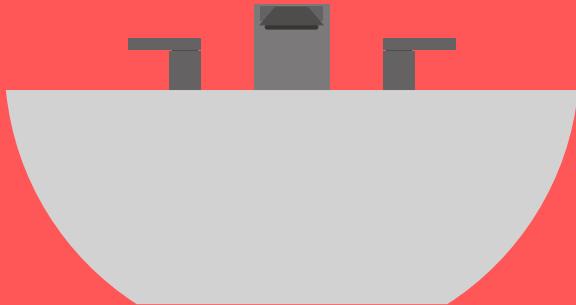
```
loadRedditPosts('/r/funny')
  //this runs if promise is resolved:
  .then((res) => {
    console.log(res.data);
  })
  //this runs if promise is rejected:
  .catch((err) => {
    console.log('Oh No!', err);
  });
}
```

This function returns a Promise which is randomly resolved/rejected.



```
const makeFakeRequest = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const randNum = Math.random();
      if (randNum > 0.5) resolve({ data: 'lol', status: 200 });
      reject({ status: 404, data: 'NO DICE' });
    }, 1000);
  });
};
```

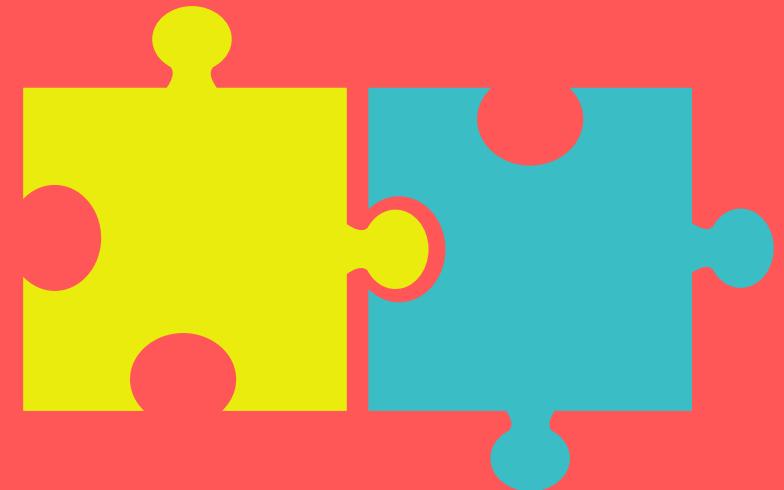
# ASYNC FUNCTIONS



A newer and cleaner syntax for  
working with async code!  
Syntax "*makeup*" for promises

# 2 PIECES

- ASYNC
- WAIT



# The `async` keyword

- Async functions always return a promise.
- If the function returns a value, the promise will be resolved with that value
- If the function throws an exception, the promise will be rejected



```
async function hello() {  
    return 'Hey guy!';  
}  
hello();  
// Promise {<resolved>: "Hey guy!"}  
async function uhOh() {  
    throw new Error('oh no!');  
}  
uhOh();  
//Promise {<rejected>: Error: oh no!}
```

# The *await* keyword

- We can only use the `await` keyword inside of functions declared with `async`.
- `await` will pause the execution of the function, **waiting for a promise to be resolved**

# REQUESTS

- XMLHTTP
- FETCH
- AXIOS



# A J A X

- ASYNCHRONOUS
- JAVASCRIPT
- AND
- XML



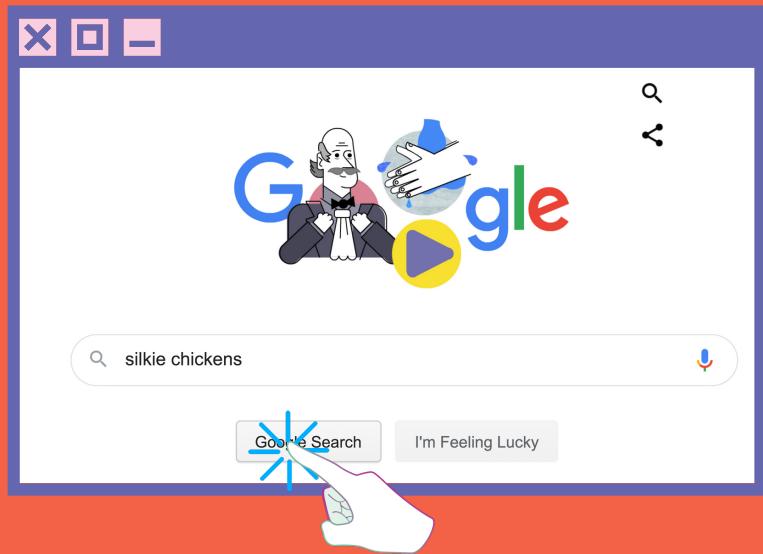


# HTTP REQUESTS

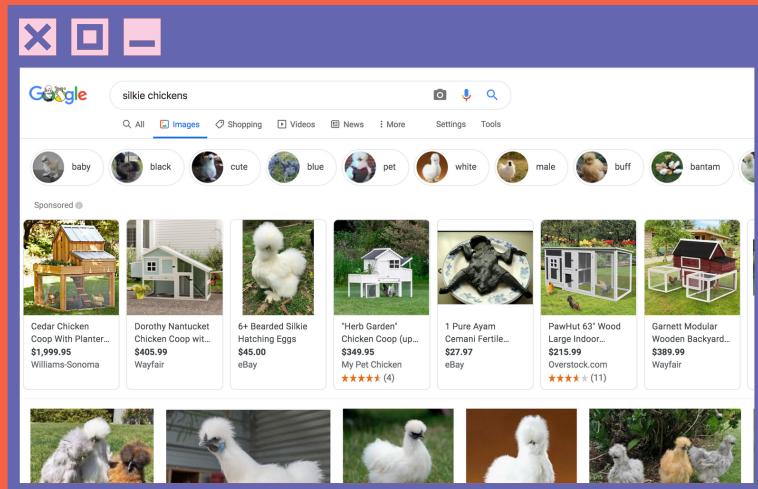
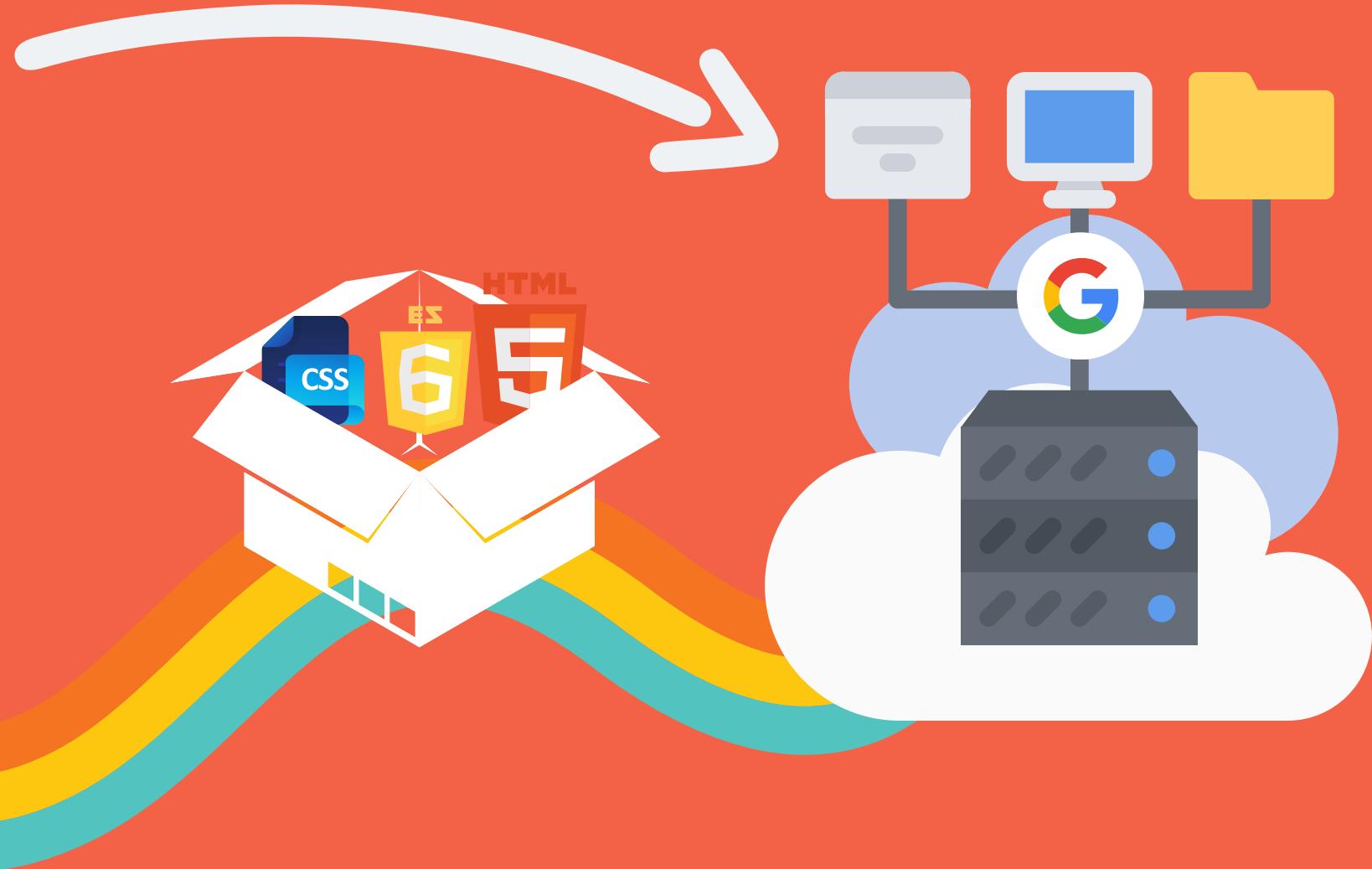


- Foundation of communication on the World Wide Web
- "Hyper Text Transfer Protocol"
- Request -> *I would like this information please*
- Response -> *Ok, here you go!*



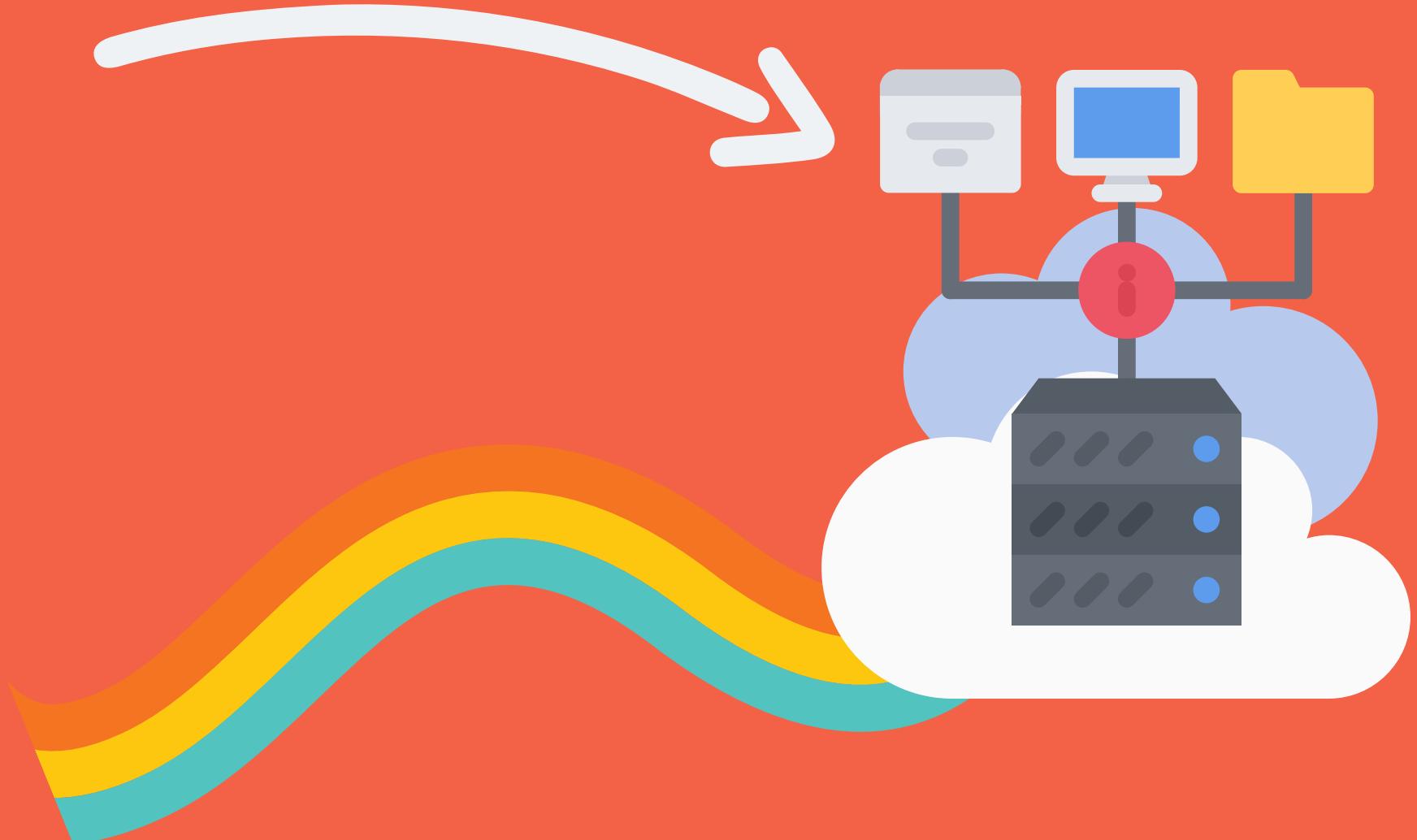


PLEASE GIVE ME  
GOOGLE.COM/SEARCH?Q=CHICKENS





PLEASE GIVE ME  
[API.CRYPTONATOR.COM/API/TICKER/BTC-USD](https://api.cryptonator.com/api/ticker/BTC-USD)



price is 11227.76

# A J A X

- ASYNCHRONOUS
- JAVASCRIPT
- AND
- XML



# AJAJ

- ASYNCHRONOUS
- JAVASCRIPT
- AND
- JSON



# JSON

- Java
- Script
- Object
- Notation

```
{  
  "squadName": "Super hero squad",  
  "homeTown": "Metro City",  
  "formed": 2016,  
  "secretBase": "Super tower",  
  "active": true,  
  "members": [  
    "Molecule Man",  
    "Madame Uppercut",  
    "Eternal Flame"  
  ]  
}
```

# POST MAN



# XMLHttpRequest

- The "original" way of sending requests via JS.
- Does not support promises, so...lots of callbacks!
- WTF is going on with the weird capitalization?
- Clunky syntax that I find difficult to remember!



# XMLHttpRequest



```
const myReq = new XMLHttpRequest();

myReq.onload = function() {
    const data = JSON.parse(this.responseText);
    console.log(data);
};

myReq.onerror = function(err) {
    console.log('ERROR!', err);
};

myReq.open('get', 'https://icanhazdadjoke.com/', true);
myReq.setRequestHeader('Accept', 'application/json');
myReq.send();
```

# Fetch API

- The newer way of making requests via JS
- Supports promises!
- Not supported in Internet Explorer :(

# FETCH



```
fetch('https://icanhazdadjoke.com/23/2', {
    headers : { Accept: 'application/json' }
})
    .then((res) => {
        if (res.status !== 200) {
            console.log('Problem!', res.status);
            return;
        }
        res.json().then((data) => {
            console.log(data);
        });
    })
    .catch(function(err) {
        console.log('Fetch Error', err);
    });
}
```

# AXIOS

## A LIBRARY FOR MAKING HTTP REQUESTS



# TERMINAL



# SETUP

## Mac

Open up the built-in terminal app

## PC

Enable Windows Subsystem  
(follow the linked tutorial)



**Develop Faster**

The terminal takes some getting used to, but it can be MUCH faster than using a GUI.



# Speed!



# Access

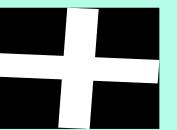
**With Great Power...**

The terminal provides a "mainline" into the heart of our computer, giving us access to areas we normally don't interact with.



# Tools!

Many of the tools we need are installed and used via the command line. We don't have much of a choice!



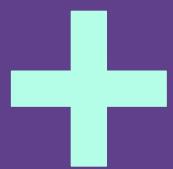
# Confusing Terminology

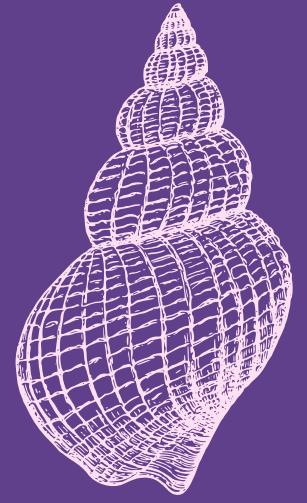
Terminal?  
Shell?  
Command Line?  
Console?  
Bash?



# Terminal

A TEXT-BASED INTERFACE TO YOUR COMPUTER.  
ORIGINALLY A PHYSICAL OBJECT, BUT NOW WE  
USE SOFTWARE TERMINALS





# shell

THE PROGRAM RUNNING ON THE TERMINAL.



# A QUICK ANALOGY!



**TERMINAL**

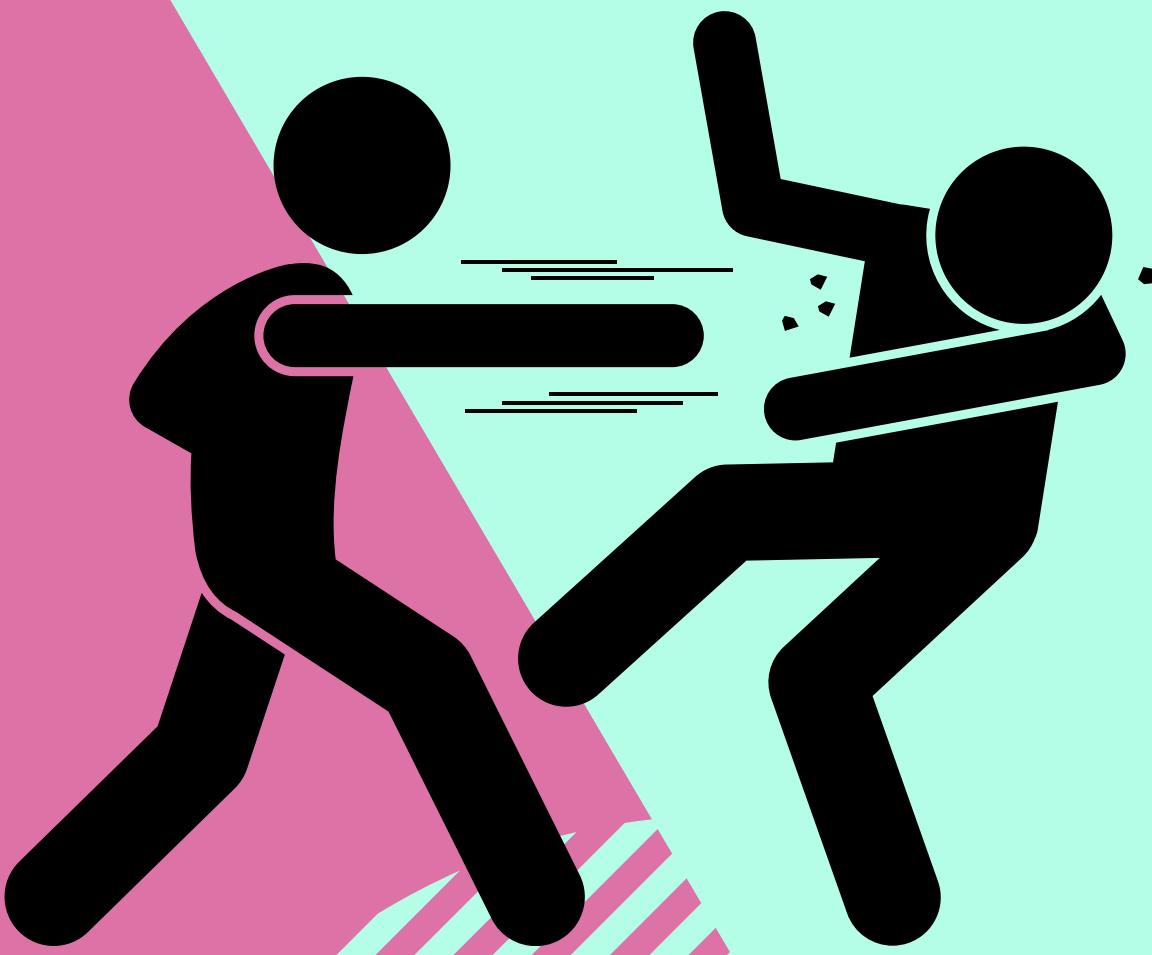
In this stupid analogy, the  
ATM is the terminal

**SHELL**

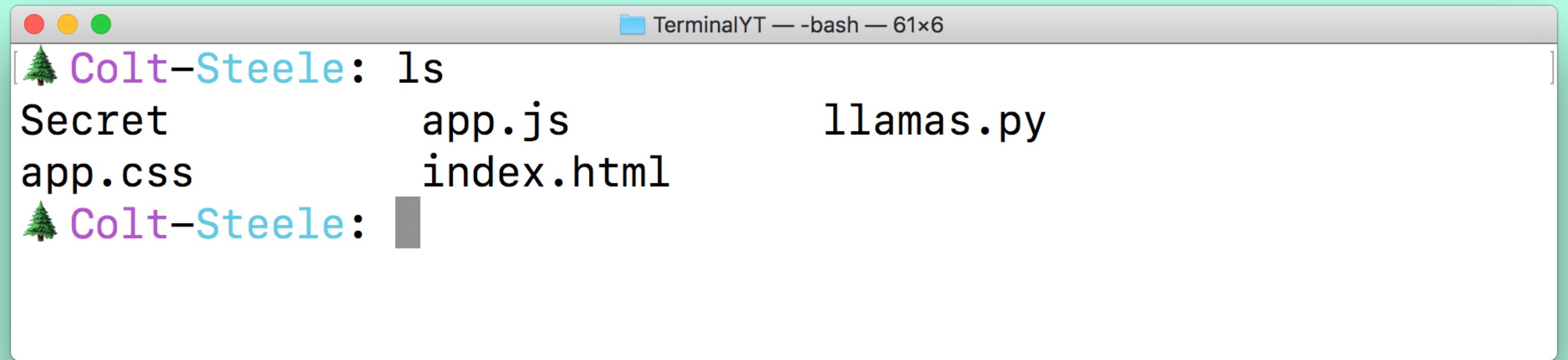
The software running on the  
ATM is the shell

# BASH

ONE OF THE MOST POPULAR SHELLS  
(AND THE DEFAULT ON A MAC)



L  
S



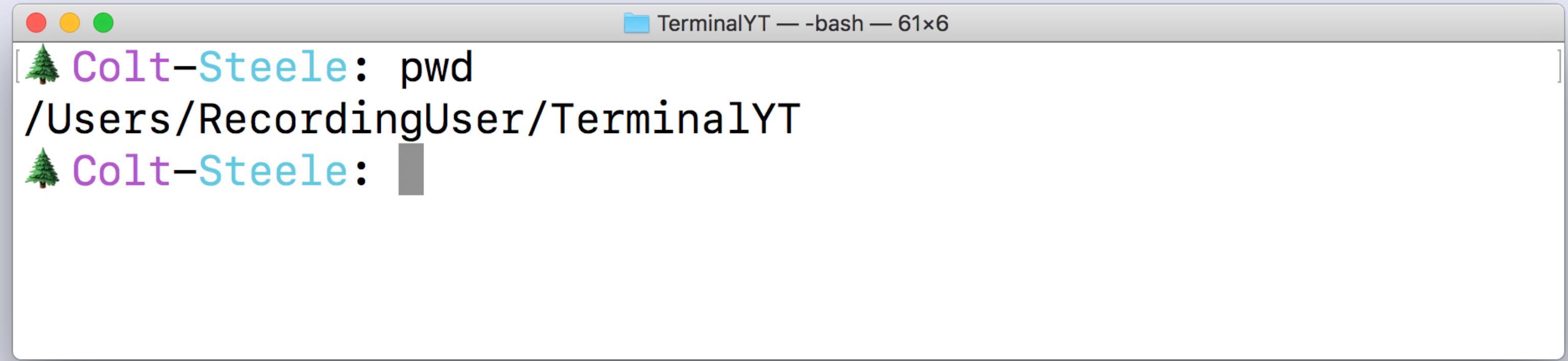
A screenshot of a macOS terminal window titled "TerminalYT — -bash — 61x6". The window shows the command "ls" being run in a directory named "Colt-Steele". The output of the command is a list of files: "Secret", "app.css", "app.js", "index.html", and "llamas.py". The terminal has a light gray background with dark gray text. The prompt "Colt-Steele:" appears twice, once before the command and once after the output.

```
[Colt-Steele: ls
Secret           app.js          llamas.py
app.css          index.html
[Colt-Steele:
```

List

Use `/s` to list the contents of your current directory

P  
W  
D



A screenshot of a macOS Terminal window titled "TerminalYT — bash — 61x6". The window shows the user's home directory: "/Users/RecordingUser/TerminalYT". The terminal prompt is "Colt-Steele:" followed by a cursor. The window has a standard OS X title bar with red, yellow, and green buttons.

```
[Colt-Steele: pwd
/Users/RecordingUser/TerminalYT
Colt-Steele: ]
```

## Print Working Directory

Prints the path to the working directory  
(where you currently are)

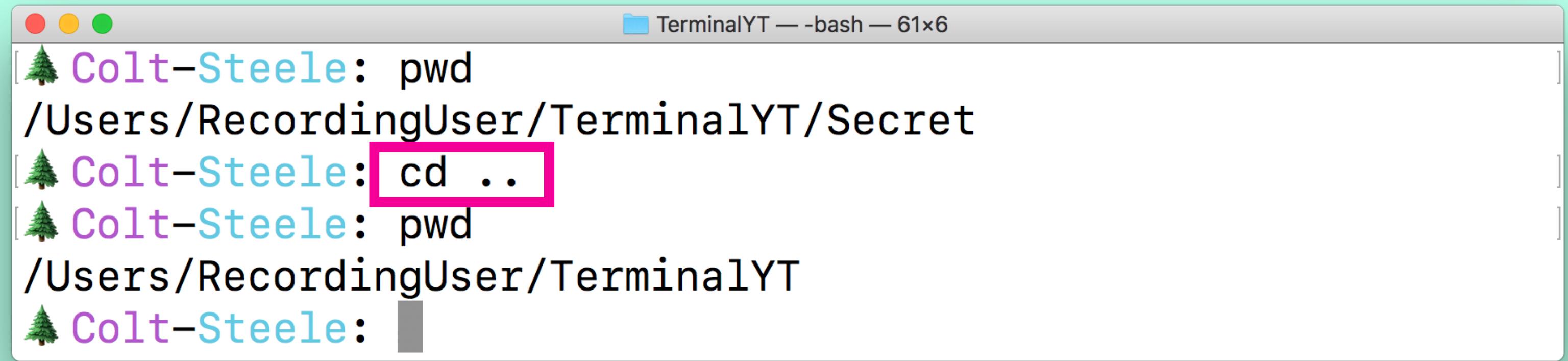
C  
D

```
[Colt-Steele: ls
Secret          app.js           llamas.py
app.css         index.html
[Colt-Steele: cd Secret/
[Colt-Steele: ls
my_diary.html  passwords.txt
[Colt-Steele:
```

## Change Directory

Use `cd` to change and move between folders

C  
D



A screenshot of a macOS Terminal window titled "TerminalYT — -bash — 61x6". The window shows the following command history:

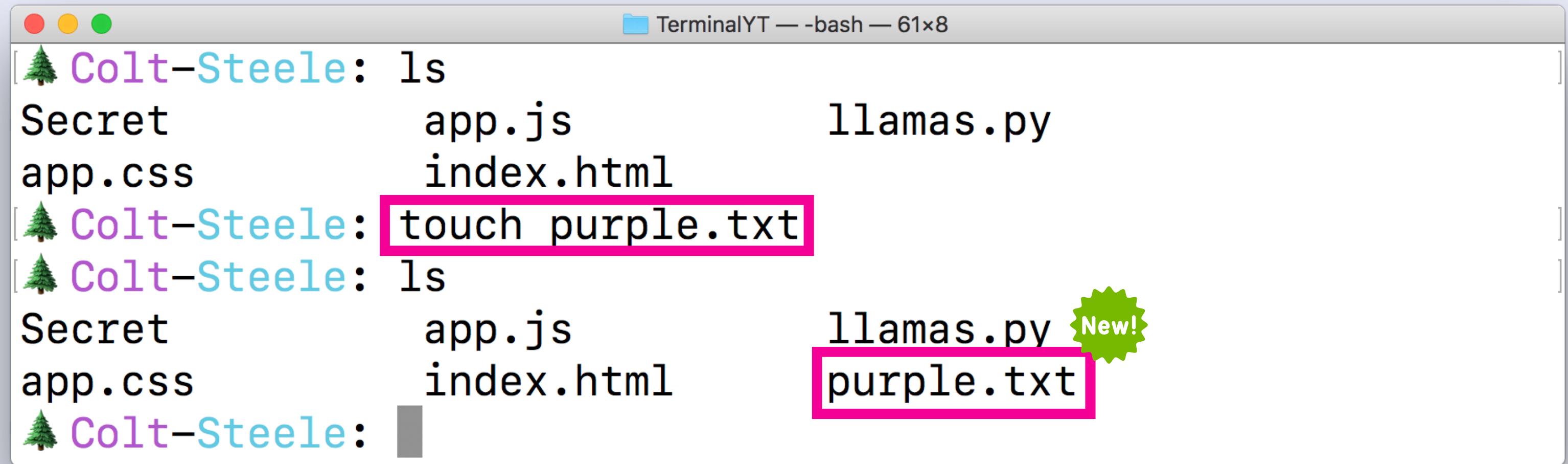
```
[Colt-Steele: pwd  
/Users/RecordingUser/TerminalYT/Secret  
[Colt-Steele: cd ..  
[Colt-Steele: pwd  
/Users/RecordingUser/TerminalYT  
Colt-Steele: ]
```

The command `cd ..` is highlighted with a pink rectangular box.

`cd ..`

Use `cd ..` to "back up" one directory

# t o u c h



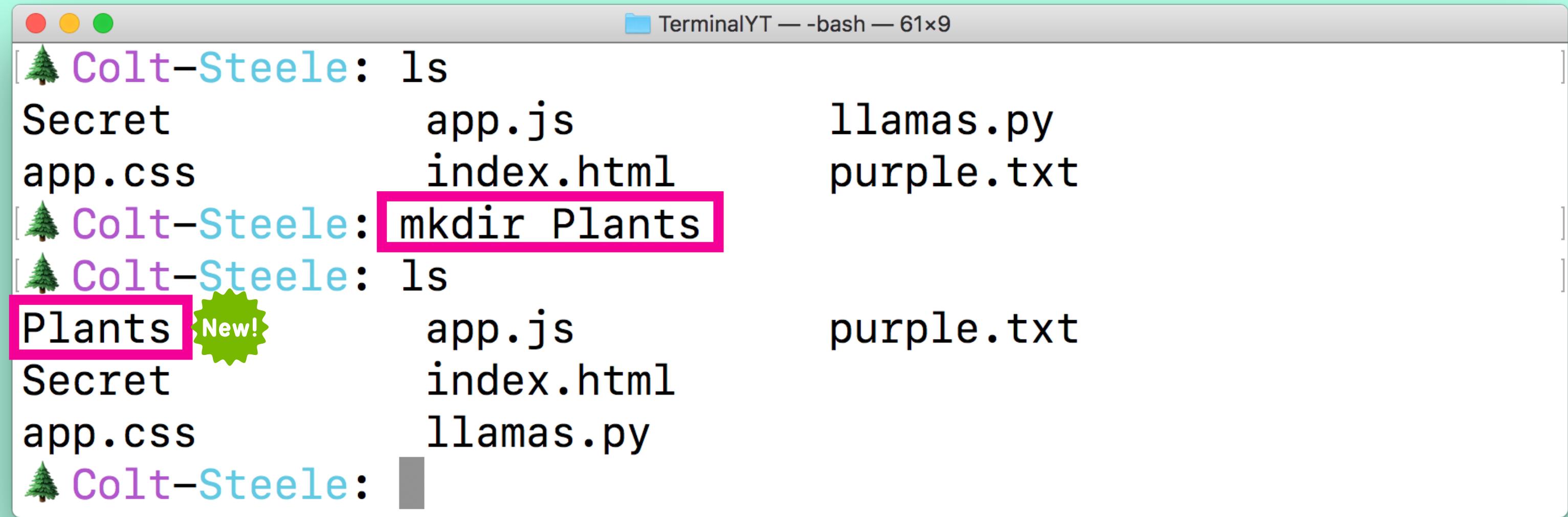
```
[Colt-Steele: ls
Secret          app.js        llamas.py
app.css         index.html
[Colt-Steele: touch purple.txt
[Colt-Steele: ls
Secret          app.js        llamas.py New!
app.css         index.html
purple.txt
[Colt-Steele:
```

The terminal window shows the user navigating to their home directory and listing files. They then run the 'touch' command to create a new file named 'purple.txt'. After creating the file, they list the files again, and the new file appears with a green 'New!' badge next to it.

## Touch

Use `touch` to create a file (or multiple)  
Yes, the name is weird...

m  
k  
d  
i  
r



A screenshot of a macOS Terminal window titled "TerminalYT — bash — 61x9". The window shows the following sequence of commands and outputs:

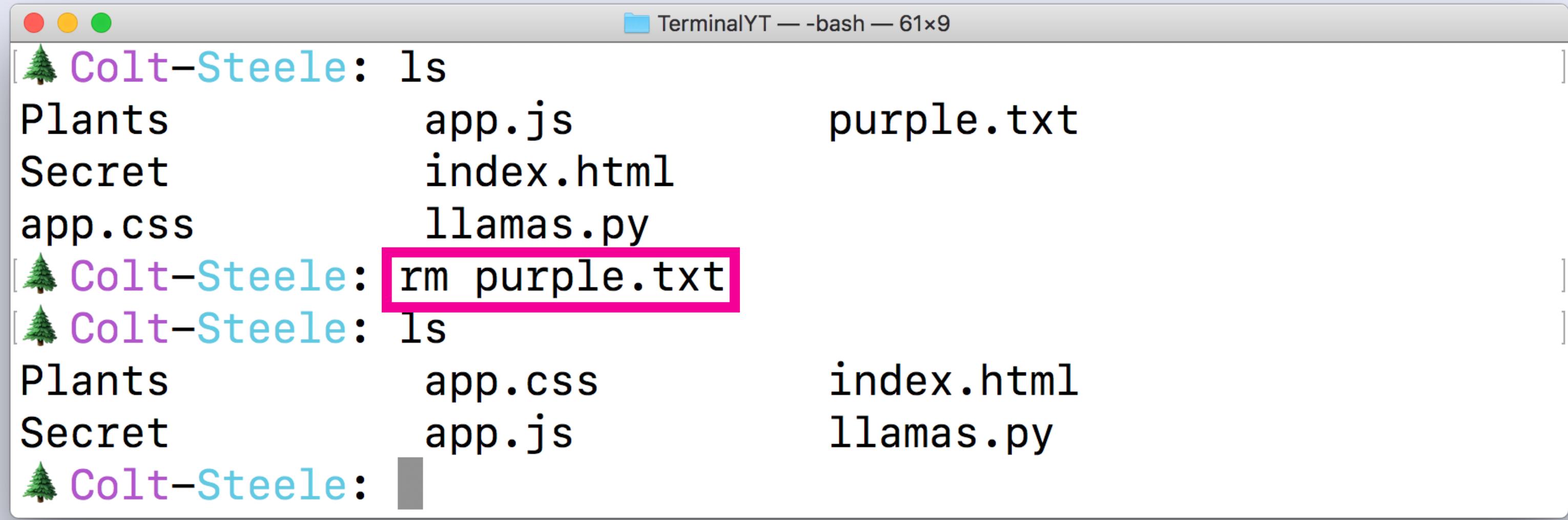
```
[Colt-Steele: ls
Secret          app.js
app.css         index.html
[Colt-Steele: mkdir Plants
[Colt-Steele: ls
Plants [New!]   app.js
Secret          index.html
app.css         llamas.py
[Colt-Steele: ]
```

The "mkdir Plants" command is highlighted with a pink rectangle. In the file list after creation, "Plants" is highlighted with a pink rectangle and has a green "New!" badge.

## mkdir (make directory)

mkdir will create a new directory  
(or directories)

rm



A screenshot of a macOS Terminal window titled "TerminalYT — bash — 61x9". The window shows the following sequence of commands and outputs:

- [Colt-Steele: ls
- Plants app.js purple.txt
- Secret index.html
- app.css llamas.py
- [Colt-Steele: rm purple.txt
- [Colt-Steele: ls
- Plants app.css index.html
- Secret app.js llamas.py
- [Colt-Steele: ]

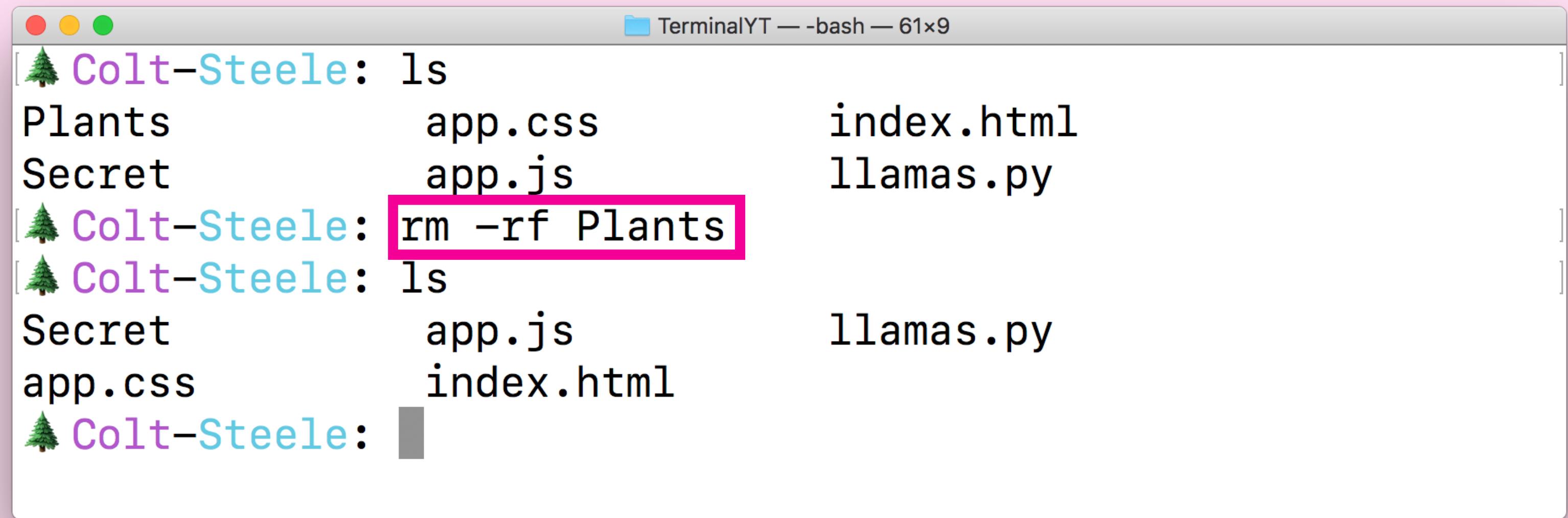
The command "rm purple.txt" is highlighted with a pink rectangle.

rm



rm will delete a file or files  
**It permanently removes them!**

r  
m



A screenshot of a macOS Terminal window titled "TerminalYT — bash — 61x9". The window shows the following session:

```
[Colt-Steele: ~] ls
Plants          app.css      index.html
Secret          app.js       llamas.py
[Colt-Steele: ~] rm -rf Plants
[Colt-Steele: ~] ls
Secret          app.js       llamas.py
app.css         index.html
[Colt-Steele: ~]
```

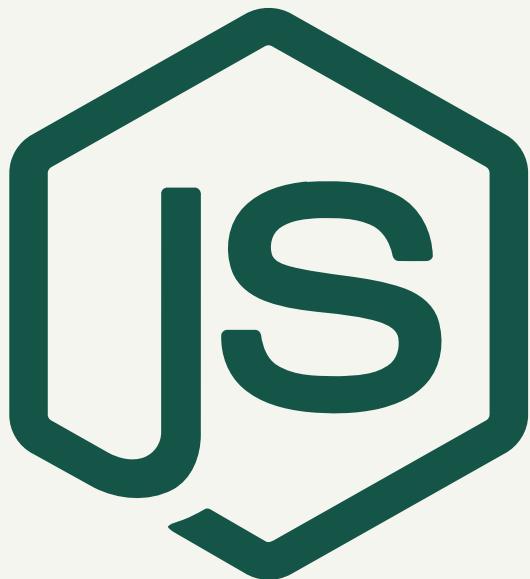
The command `rm -rf Plants` is highlighted with a pink rectangle.

**rm -rf**

use `rm -rf` to delete a directory  
(r = recursive, f = force)



WEB DEVELOPER BOOTCAMP



## WHAT IS NODE?

"A JAVASCRIPT RUNTIME"

---

Until recently, we could only run JavaScript code in a web browser. Node is a JavaScript runtime that executes code outside of the browser.

We can use the same JavaScript syntax we know and love to write server-side code, instead of relying on other languages like Python or Ruby.



# WHAT DO PEOPLE BUILD WITH IT?

- Web Servers
- Command Line Tools
- Native Apps (VSCode is a Node app!)
- Video Games
- Drone Software
- A Whole Lot More!

# NODE JS VS CLIENT-SIDE JS

## NOT INCLUDED IN NODE

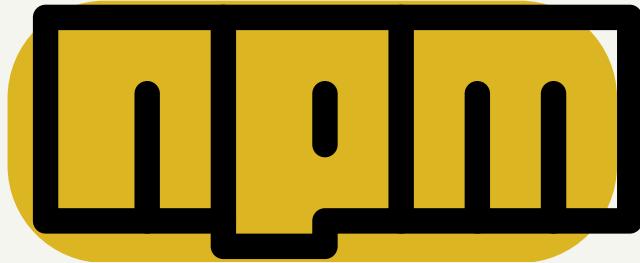


Because Node does not run in a browser, we don't have access to all the browser "stuff". The window, document, and DOM API's are not a thing in Node!

## NEW STUFF IN NODE!



Node comes with a bunch of built-in modules that don't exist in the browser. These modules help us do things like interact with the operating system and files/folders.



## NPM

### NODE PACKAGE MANAGER

---

NPM is really two things:

1. A library of thousands of packages published by other developers that we can use for free!
2. A command line tool to easily install and manage those packages in our Node projects

# express

WEB DEVELOPER BOOTCAMP



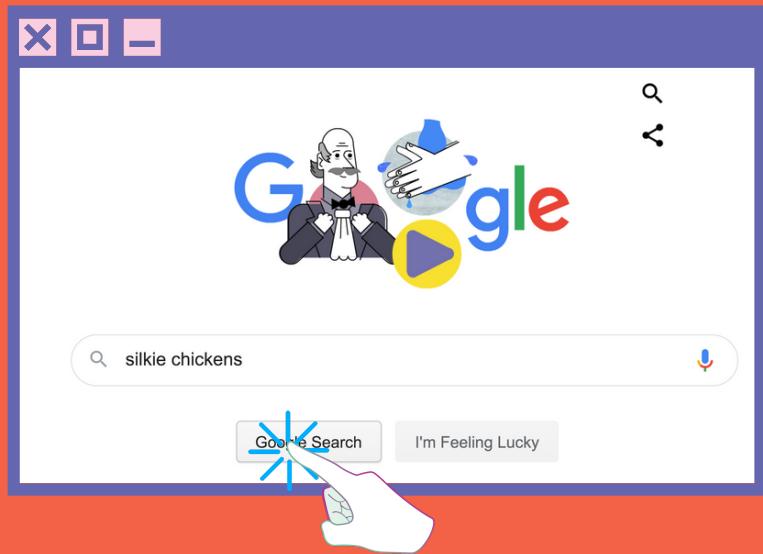
## WHAT IS EXPRESS?

### OUR FIRST FRAMEWORK!

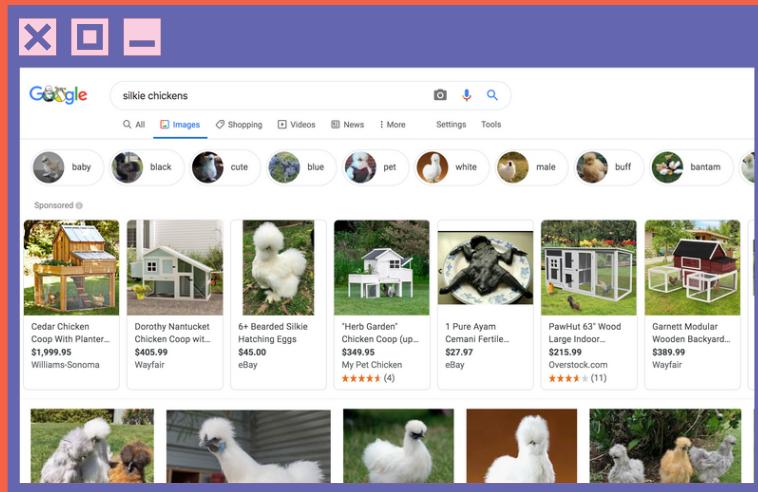
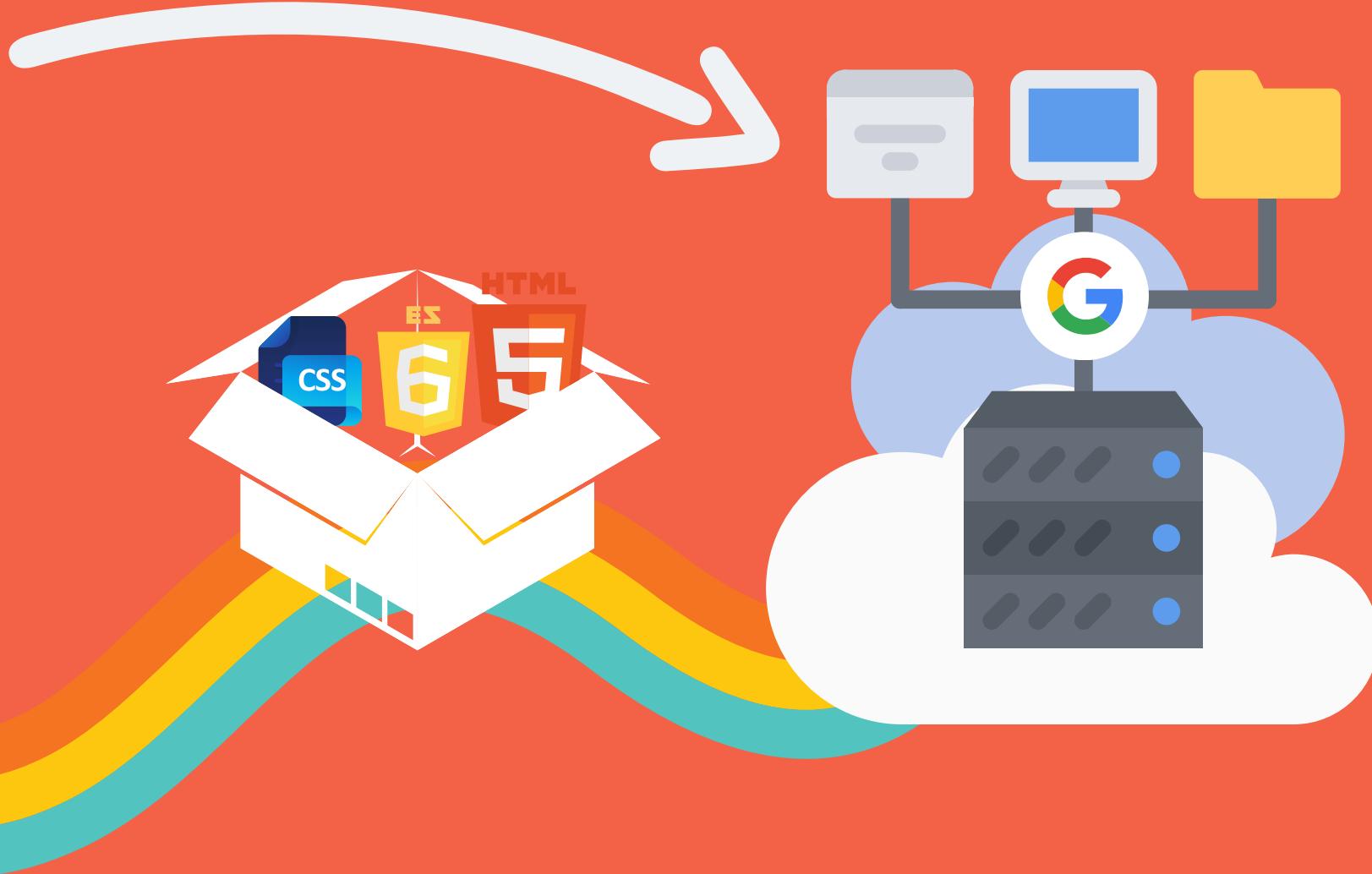
---

Express is a "fast, unopinionated, minimalist web framework for Node.js:" It helps us build web apps!

It's just an NPM package which comes with a bunch of methods and optional plugins that we can use to build web applications and API's



PLEASE GIVE ME  
GOOGLE.COM/SEARCH?Q=CHICKENS





## EXPRESS HELPS US...

- Start up a server to listen for requests
- Parse incoming requests
- Match those requests to particular routes
- Craft our http response and associated content

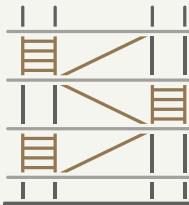
# LIBRARIES VS. FRAMEWORKS

## LIBRARY



When you use a library, you are in charge!  
You control the flow of the application  
code, and you decide when to use the  
library.

## FRAMEWORK

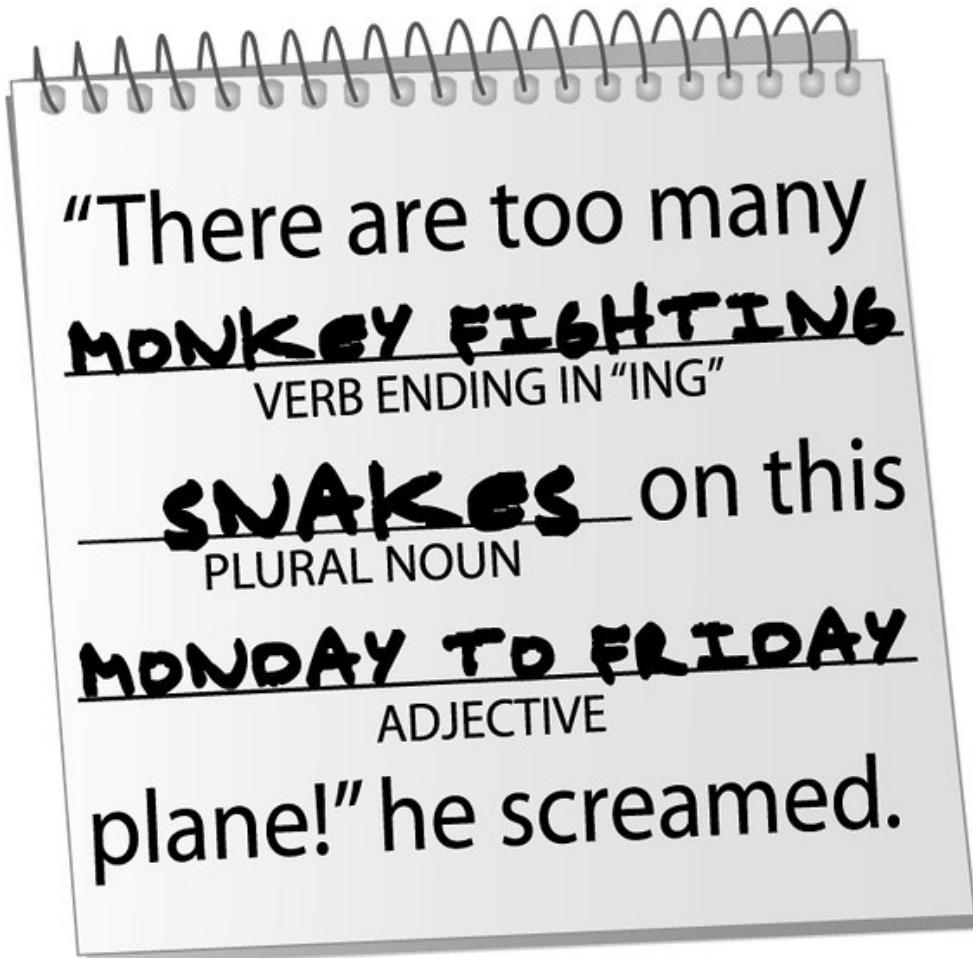


With frameworks, that control is inverted.  
The framework is in charge, and you are  
merely a participant! The framework tells  
you where to plug in the code.

## TEMPLATING

Templating allows us to define a preset "pattern" for a webpage, that we can dynamically modify.

For example, we could define a single "Search" template that displays all the results for a given search term. We don't know what the term is or how many results there are ahead of time. The webpage is created on the fly.



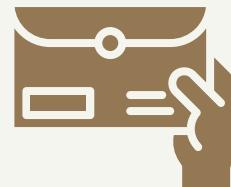
## GET VS. POST

GET



- Used to retrieve information
- Data is sent via query string
- Information is plainly visible in the URL!
- Limited amount of data can be sent

POST



- Used to post data to the server
- Used to write/create/update
- Data is sent via request body, not a query string!
- Can send any sort of data (JSON!)

# REST

WEB DEVELOPER BOOTCAMP



## WTF IS REST?

REPRESENTATIONAL  
STATE TRANSFER

---

REST is an "architectural style for distributed hypermedia systems". Yikes.

It's basically a set of guidelines for how a client + server should communicate and perform CRUD operations on a given resource



## WTF IS REST?

REPRESENTATIONAL  
STATE TRANSFER

---

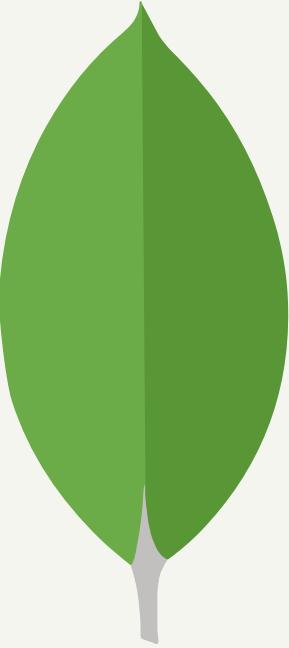
The main idea of REST is treating data on the server-side as resources than can be CRUDed

The most common way of approaching REST is in formatting the URLs and HTTP verbs in your applications.

# AN EXAMPLE W/ COMMENTS

THIS IS JUST ONE APPROACH TO RESTFUL ROUTING:

NAME	PATH	VERB	PURPOSE
Index	/comments	GET	Display all comments
New	/comments/new	GET	Form to create new comment
Create	/comments	POST	Creates new comment on server
Show	/comments/:id	GET	Details for one specific comment
Edit	/comments/:id/edit	GET	Form to edit specific comment
Update	/comments/:id	PATCH	Updates specific comment on server
Destroy	/comments/:id	DELETE	Deletes specific item on server



# mongoDB

WEB DEVELOPER BOOTCAMP





## WHAT IS MONGO?

### OUR FIRST DATABASE!

---

According to Mongo's homepage, it is "the most popular database for modern applications". It is commonly used in combination with Node.

Mongo is a document database, which we can use to store and retrieve complex data from.



## WHY USE A DATABASE?

INSTEAD OF JUST SAVING TO A FILE?

- Databases can handle large amounts of data efficiently and store it compactly
- They provide tools for easy insertion, querying, and updating of data
- They generally offer security features and control over access to data
- They (generally) scale well.

# SQL VS. NOSQL

## SQL DATABASES



Structured Query Language databases are relational databases. We pre-define a schema of tables before we insert anything.

## NO-SQL DATABASES



NoSQL databases do not use SQL. There are many types of no-sql databases, including document, key-value, and graph stores.

# POPULAR DATABASES

## SQL DATABASES

- MySQL
- Postgres
- SQLite
- Oracle
- Microsoft SQL Server

## NO-SQL DATABASES

- MongoDB
- Couch DB
- Neo4j
- Cassandra
- Redis



## WHY ARE WE LEARNING MONGO?

- Mongo is very commonly used with Node and Express (MEAN & MERN stacks)
- It's easy to get started with (though it can be tricky to truly master)
- It plays particularly well with JavaScript
- Its popularity also means there is a strong community of developers using Mongo.

# Express + Mongoose

WEB DEVELOPER BOOTCAMP



mongoDB



JS

## ODM

OBJECT DATA MAPPER?

OBJECT DOCUMENT MAPPER?

ODMs like Mongoose map documents coming from a database into usable JavaScript objects.

Mongoose provides ways for us to model out our application data and define a schema. It offers easy ways to validate data and build complex queries from the comfort of JS.

# MIDDLEWARE

**REQUEST**  
→



**RESPONSE**  
→

Express middleware are functions that run during the request/response lifecycle.

# MIDDLEWARE

**R E Q U E S T**  
→

- Middleware are just functions
- Each middleware has access to the request and response objects
- Middleware can end the HTTP request by sending back a response with methods like `res.send()`
- OR middleware can be chained together, one after another by calling `next()`

**R E S P O N S E**  
→

# ERRORS!

REQUEST  
→



RESPONSE  
→

How do we handle errors in an Express application?



# MONGO RELATIONSHIPS

# ONE TO FEW

Embed the data directly in the document!

```
{  
  name: 'Tommy Cash',  
  savedAddresses : [  
    { street: 'Rahukohtu 3', city: 'Tallinn ', country: 'Estonia' },  
    { street: 'Rävala 5', city: 'Tallinn ', country: 'Estonia' }  
  ]  
}
```

# ONE TO MANY

One option is to store your data separately, but then store references to document ID's somewhere inside the parent:

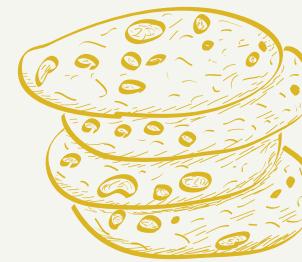
```
{  
  farmName: 'Full Belly Farms',  
  location: 'Guinda, CA',  
  produce : [  
    ObjectId('2819781267781'),  
    ObjectId('1828678675667'),  
    ObjectId('8187777231283'),  
  ]  
}
```

# ONE TO BAJILLIONS

With thousands or more documents, it's more efficient to store a reference to the parent on the child document.

```
{  
  tweetText: 'lol I just crashed my car because I was tweeting',  
  tags: ['stupid', 'moron', 'yolo'],  
  user: ObjectId('2133243243')  
}
```

# COOKIES



WEB DEVELOPER  
BOOTCAMP



# COOKIES

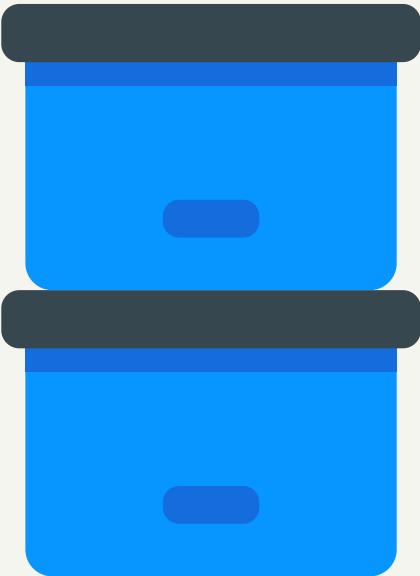
## WHAT ARE THEY?

---

Cookies are little bits of information that are stored in a user's browser when browsing a particular website.

Once a cookie is set, a user's browser will send the cookie on every subsequent request to the site.

Cookies allow use to make HTTP stateful



# SESSIONS

## WHAT ARE THEY?

---

It's not very practical (or secure) to store a lot of data client-side using cookies. This is where sessions come in!

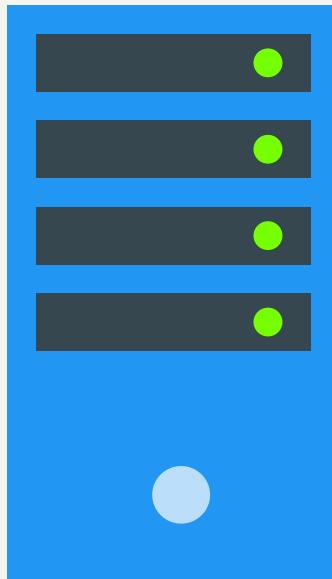
Sessions are a server-side data store that we use to make HTTP stateful. Instead of storing data using cookies, we store the data on the server-side and then send the browser a cookie that can be used to retrieve the data .

A diagram might be helpful here.

# DATA STORE

```
{  
  id: 3,  
  shoppingCart: [  
    {item: 'lime', qty:1},  
    {item: 'la croix', qty:99},  
    {item: 'lemon', qty:2},  
  ]  
},  
{  
  id: 4,  
  shoppingCart: [  
    {item: 'carrot', qty:2},  
    {item: 'celery', qty:5},  
    {item: 'taser;', qty:99},  
  ]  
},  
{  
  id: 5,  
  shoppingCart: [  
    {item: 'apple', qty:2},  
    {item: 'onion', qty:5},  
    {item: 'pear;', qty:9},  
  ]  
}
```

## SERVER

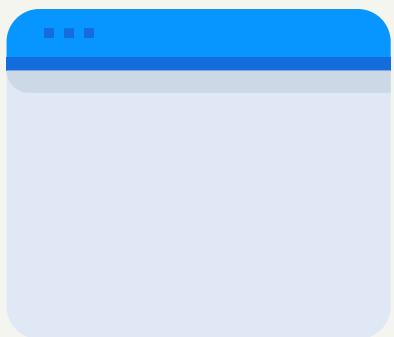


Your session ID is 4

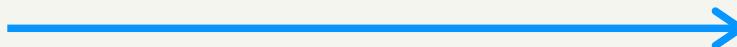
## CLIENT



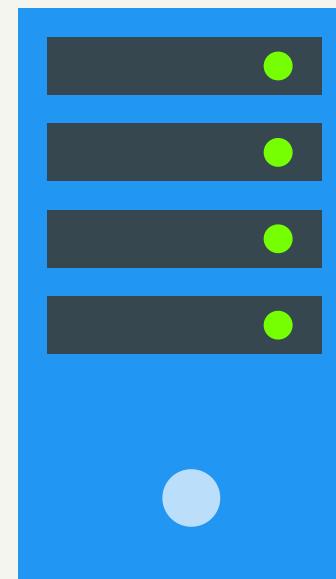
## CLIENT



I have a cookie for you!  
Session ID is 4



## SERVER



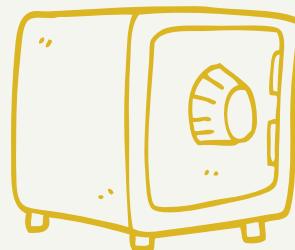
## DATA STORE

```
{  
  id: 4,  
  shoppingCart: [  
    {item: 'carrot', qty:2},  
    {item: 'celery', qty:5},  
    {item: 'taser;', qty:99},  
  ]  
}
```

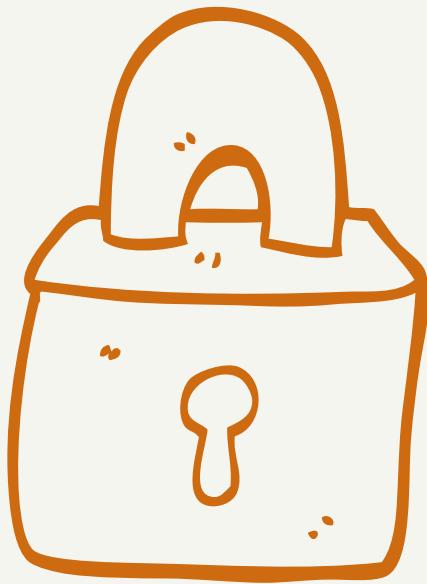
DEEP BREATH. IT'S TIME FOR AUTH.

DEEP BREATH. IT'S TIME FOR AUTH.

# A U T H



WEB DEVELOPER  
BOOTCAMP



## AUTHENTICATION

### WHAT IS IT?

---

Authentication is the process of verifying who a particular user is.

We typically authenticate with a username/password combo, but we can also use security questions, facial recognition, etc.



## AUTHORIZATION

### WHAT IS IT?

---

Authorization is verifying what a specific user has access to.

Generally, we authorize after a user has been authenticated.  
"Now that we know who you are, here is what you are  
allowed to do and NOT allowed to do"

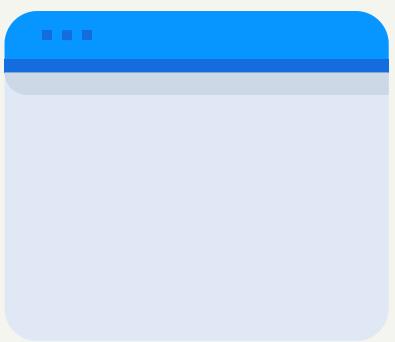
# R U L E # 1

## NEVER STORE PASSWORDS

# R U L E # 1

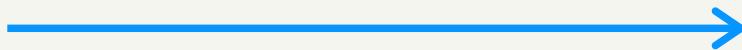
## NEVER STORE PASSWORDS

```
{  
    username: 'kittycatluvr',  
    password: 'meowmeow999'  
},  
{  
    username: 'geckoGuy',  
    password: 'lizard987'  
}
```



## CLIENT

LOG ME IN WITH:  
Username: 'geckoGuy'  
Password: 'lizard987'



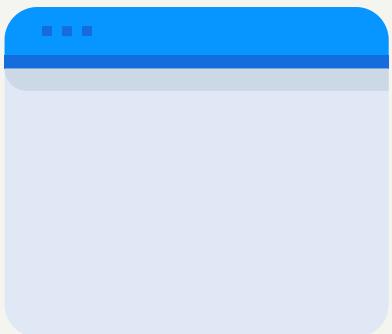
## SERVER



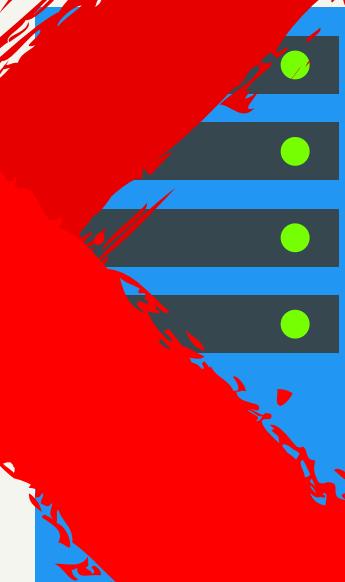
## DATABASE

```
{  
    username: 'kittycatluvr',  
    password: 'meowmeow999'  
},  
{  
    username: 'geckoGuy',  
    password: 'lizard987'  
}
```

## CLIENT



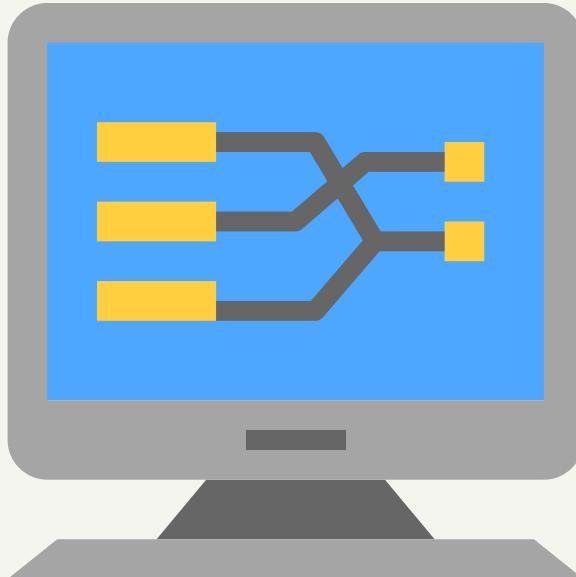
LOG ME IN  
Username: 'geckoGuy'  
Password: 'lizard987'



## DATABASE

```
{  
  username: 'kittycatluvr',  
  password: 'meowmeow999!'  
},  
{
```

```
  username: 'geckoGuy',  
  password: 'lizard987'  
}
```



## HASHING

### THE SOLUTION!

---

Rather than storing a password in the database, we run the password through a hashing function first and then store the result in the database.

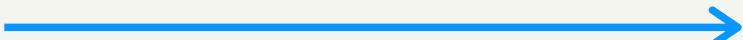
# HASHING FUNCTIONS

Hashing functions are functions that map input data of some arbitrary size to fixed-size output values.

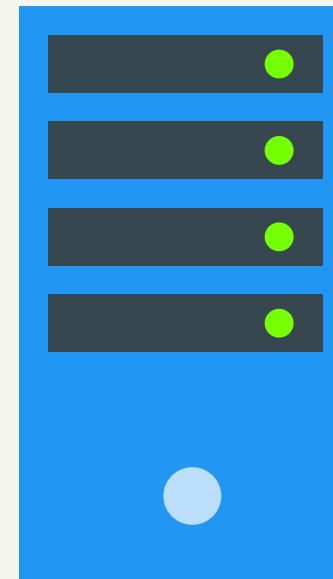


# CLIENT

LOG ME IN WITH:  
Username: 'geckoGuy'  
Password: 'lizard987'



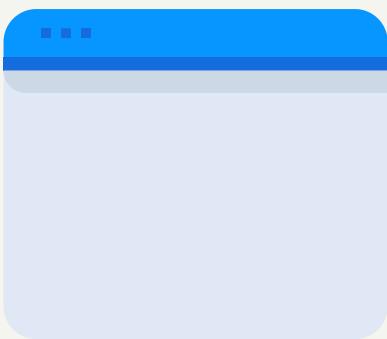
# SERVER



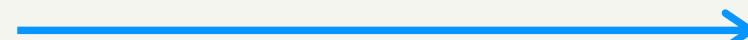
# DATABASE

```
{  
    username: 'kittycatluvr',  
    password:'d7offoab9a23ec5dba9075boe4de  
    de8c2972ba933d6d5adf3a42abb6eod7a2da'  
},  
{  
    username: 'geckoGuy',  
    password:'07123e1f482356c415f684407a3b87  
    23e1ob2cbbcob8fcfd6282c49d37c9ciabc'  
}
```

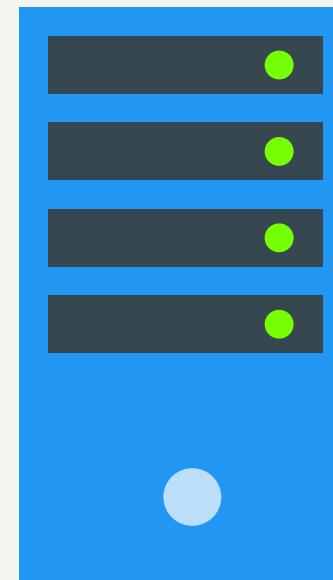
## CLIENT



LOG ME IN WITH:  
Username: 'geckoGuy'  
Password: 'lizard987'

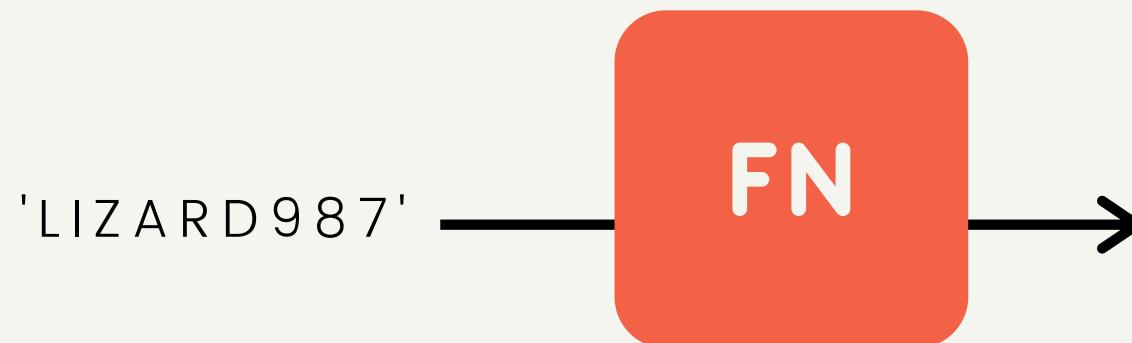


## SERVER

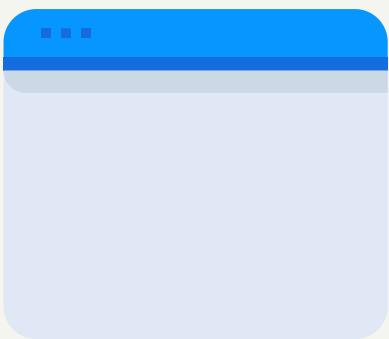


## DATABASE

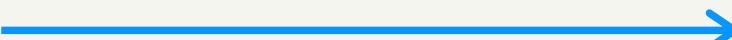
```
{  
    username: 'kittycatluvr',  
    password:'d7offoab9a23ec5dba9075boe4de  
    de8c2972ba933d6d5adf3a42abb6eod7a2da'  
},  
{  
    username: 'geckoGuy',  
    password:'07123e1f482356c415f684407a3b87  
    23e1ob2cbbcob8fcfd6282c49d37c9ciabc'  
}
```



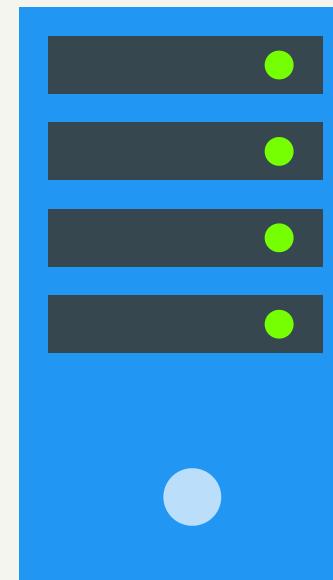
## CLIENT



LOG ME IN WITH:  
Username: 'geckoGuy'  
Password: 'lizard987'

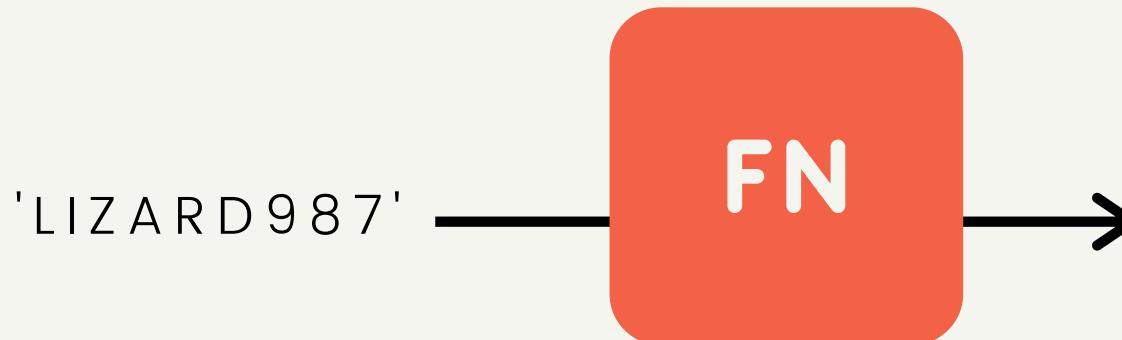


## SERVER



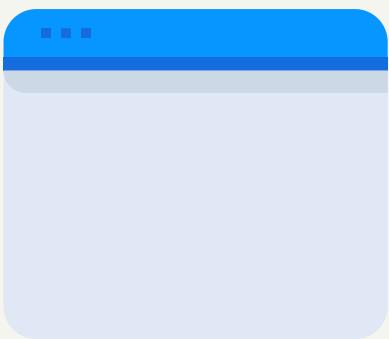
## DATABASE

```
{  
    username: 'kittycatluvr',  
    password:'d7offoab9a23ec5dba9075boe4de  
    de8c2972ba933d6d5adf3a42abb6eod7a2da'  
},  
{  
    username: 'geckoGuy',  
    password:'07123e1f482356c415f684407a3b87  
    23e1ob2cbbcob8fcfd6282c49d37c9c1abc'  
}
```

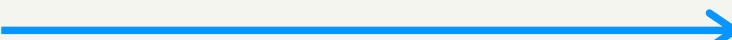


07123E1F482356C415F6844  
07A3B8723E10B2CBBC0B8F  
CD6282C49D37C9C1ABC

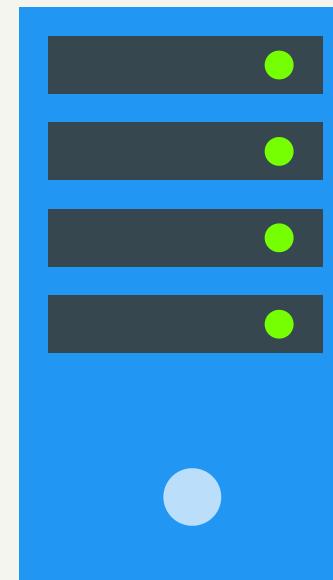
## CLIENT



LOG ME IN WITH:  
Username: 'geckoGuy'  
Password: 'lizard987'

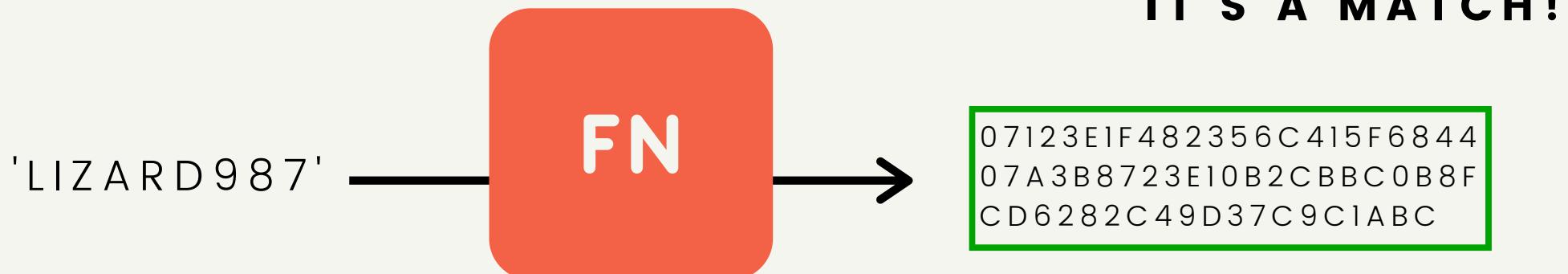


## SERVER



## DATABASE

```
{  
    username: 'kittycatluvr',  
    password:'d7offoab9a23ec5dba9075boe4de  
    de8c2972ba933d6d5adf3a42abb6eod7a2da'  
},  
{  
    username: 'geckoGuy',  
    password:'07123e1f482356c415f684407a3b87  
    23e1ob2cbbcob8fcd6282c49d37c9c1abc'  
}
```

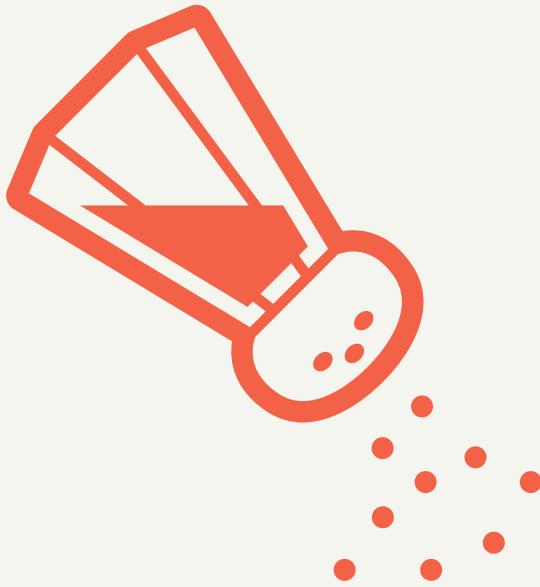


IT'S A MATCH!

# CRYPTOGRAPHIC HASH FUNCTIONS

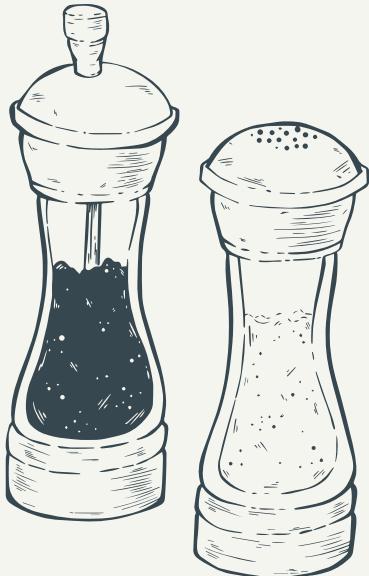
---

1. One-way function which is infeasible to invert
2. Small change in input yields large change in the output
3. Deterministic - same input yields same output
4. Unlikely to find 2 outputs with same value
5. Password Hash Functions are deliberately SLOW



# S A L T S

## AN EXTRA SAFEGUARD



## PASSWORD SALTS

OMG THAT'S SO RANDOM!

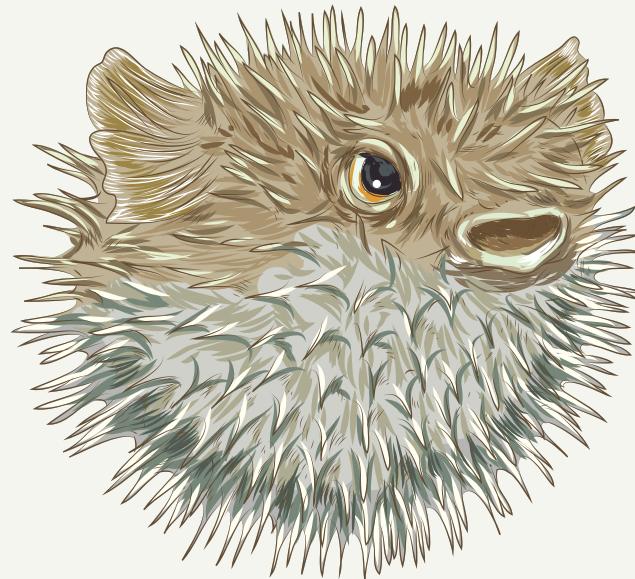
---

A salt is a random value added to the password before we hash it.

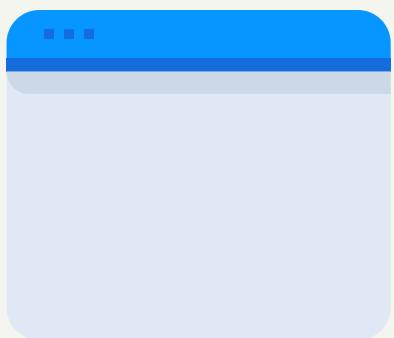
It helps ensure unique hashes and mitigate common attacks

# B C R Y P T

## OUR HASH FUNCTION!

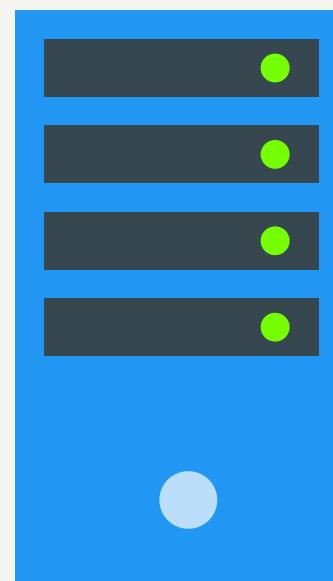


## CLIENT



I have a cookie for you!  
Session ID is 4

## SERVER



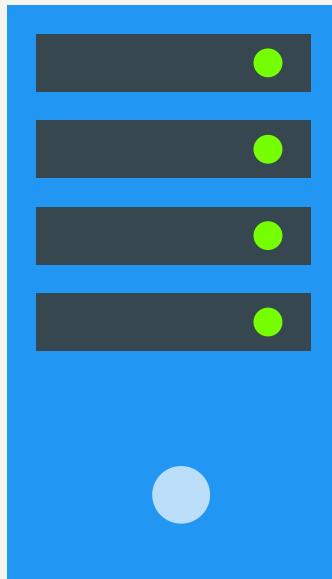
## DATA STORE

```
{  
  id: 4,  
  shoppingCart: [  
    {item: 'carrot', qty:2},  
    {item: 'celery', qty:5},  
    {item: 'taser;', qty:99},  
  ]  
}
```

# DATA STORE

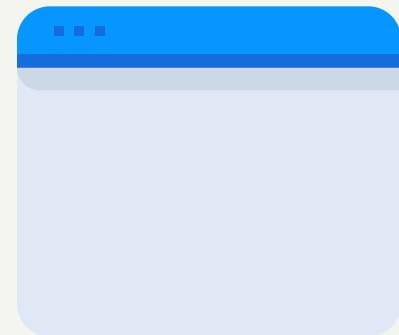
```
{  
  id: 3,  
  shoppingCart: [  
    {item: 'lime', qty:1},  
    {item: 'la croix', qty:99},  
    {item: 'lemon', qty:2},  
  ]  
},  
{  
  id: 4,  
  shoppingCart: [  
    {item: 'carrot', qty:2},  
    {item: 'celery', qty:5},  
    {item: 'taser;', qty:99},  
  ]  
},  
{  
  id: 5,  
  shoppingCart: [  
    {item: 'apple', qty:2},  
    {item: 'onion', qty:5},  
    {item: 'pear;', qty:9},  
  ]  
}
```

## SERVER



Your session ID is 4

## CLIENT



# </> HTML

## CheatSheet



In this Cheatsheet, we will cover the basics of HTML tags, elements, and attributes. We will provide examples to help you understand how these elements work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.



HTML (Hypertext Markup Language) is a standard markup language used to create web pages. It is used to structure and format content on the web, including text, images, and other multimedia elements. HTML consists of a series of elements that are represented by tags, which are used to define the structure and content of a webpage.

HTML is an essential part of the web development process and is used to create the structure and content of websites. It is a fundamental skill for web developers and is used to create the majority of websites on the internet.

## HTML COMPONENTS

- **<html> tag:** This tag acts as a container for every other element in the document except the <!DOCTYPE html> tag.
- **<head> tag:** Includes all the document's metadata.
- **<title> tag:** Defines the title of the document which is displayed in the browser's title bar.
- **<body> tag:** Acts as a container for the document's content that gets displayed on the browser.

## This is how it all comes together:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title> Code Help HTML Cheat Sheet </title>
  </head>
  <body> .... </body>
</html>
```

**<!DOCTYPE html>** specifies that we are working with an HTML5 document.

**The following tags contribute extra information to the HTML document:**

- **<meta> tag:** This tag can be used to define additional information about the webpage.
- **<link> tag:** Used to link the document to an external resource.
- **<style> tag:** Used for defining styles for the document.
- **<script> tag:** Used to write code snippets (usually JavaScript) or to link the document to an external script.

```
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Code Help HTML Cheat Sheet</title>

<style>
*{
  font-size: 40px;
}
</style>

<script>
  alert ('message');
</script>

</head>
```

## STRUCTURE OF A HTML DOCUMENT

While constructing your HTML document, you can use certain tags to establish its structure. The **<h1>** to **<h5>** tags signify different heading levels, with **<h1>** being the highest level and **<h5>** being the lowest level.

```
<h1> Heading 1 </h1>
<h2> Heading 2 </h2>
<h3> Heading 3 </h3>
<h4> Heading 4 </h4>
<h5> Heading 5 </h5>
```

You use the **<p>** tag to create a paragraph.

```
<p> This is a paragraph </p>
```

The **<div>** tag can be employed to segment and style different areas of the document. It also acts as a parent container for other elements within the document.

This is how it works:

```
<div class="About Us">
    <h1> This is the About Us section </h1>
    <p> Welcome to the About Us section! </p>
</div>

<div class="Contact Us">
    <h1> This is the Contact Us section </h1>
    <p> Contact us on 0123456789 </p>
</div>
```

We also have the **<span>** tag. This is similar to **<div>** but you use it as an inline container.

```
<p> Hello <span class="span1"> World! </span></p>
```

There's the **<br/>** tag, which we use to insert line breaks in the document. This tag does not require a closing tag.

```
<p> Welcome to <br/> Code Help </p>
```

The **<hr/>** tag is used to create a horizontal line. It also has no closing tag.

```
<p> Welcome to <hr/> Code Help </p>
```

## IMAGES IN HTML

In HTML, we use the **<img/>** tag to insert images into the document.

Here are some attributes of the **<img/>** tag.

- **src** is used to specify the location of the image on your computer or the internet.
- **alt** specifies alternative text that displays if the image cannot be rendered. This text is also helpful for screen readers.
- **height** determines the height of the image.
- **width** determines the width of the image.

- **border** specifies the thickness of the borders around the image. If no border is added, it is set to 0.

```

```

## TEXT FORMATING IN HTML

**HTML provides multiple methods for formatting text. Let's take a brief look at them now:**

- The **<i>** tag formats text in italics.
- The **<b>** tag formats text in bold.
- The **<strong>** tag also formats text in bold and is used to emphasize important information.
- The **<em>** tag is another emphasis tag that formats text in italics.
- The **<sub>** tag formats text as subscript, like Carbon Dioxide CO<sub>2</sub>.
- The **<sup>** tag formats text as a superscript, like the power of a number, 2<sup>32</sup>.
- The **<small>** tag decreases the size of text.
- The **<del>** tag formats text as deleted by striking a line through it.
- The **<address>** tag is used to show the author's contact information.
- The **<abbr>** tag denotes an abbreviation.
- The **<code>** tag formats text as code snippets.
- The **<mark>** tag highlights text.
- The **<ins>** tag formats text as inserted, which is usually underlined.

- The **<blockquote>** tag is used to enclose a section of text quoted from another source.
- The **<q>** tag is used for shorter inline quotes.
- The **<cite>** tag is used to cite the author of a quote.

```
<p><i> Italic </i></p>
<p><b> Bold </b></p>
<p><strong> Strong </strong></p>
<p><em> Strong </em></p>
<p><sub> Subscript </sub></p>
<p><sup> Superscript </sup></p>
<p><small> Small </small></p>
<p><del> Delete </del></p>
<p><address> Address </address></p>
<p><abbr> Inserted Abbreviation </abbr></p>
<p><code> Code Snippet </code></p>
<p><mark> Marked Text </mark></p>
<p><ins> Insert </ins></p>
<p><blockquote> Quoted </blockquote></p>
<p><q> Short Quoted </q></p>
<p><cite> Cited </cite></p>
```

## LINKS IN HTML

The **<a>** tag, also referred to as the anchor tag, is used to establish hyperlinks that link to other pages (external web pages included) or to a particular section within the same page.

Here are some attributes of the **<a>** tag:

- The **href** attribute specifies the URL that the link will take the user to when clicked.
- The **download** attribute specifies that the target or resource clicked is a downloadable file.
- The **target** attribute specifies where the linked document or resource should be opened. This could be in the same window or a new window.

```
<a href="https://www.thecodehelp.in/" target="_blank"> Code Help </a>
```

## LISTS IN HTML

- The **<ol>** tag defines an ordered list.
- The **<ul>** tag defines an unordered list.
- The **<li>** tag is used to create items in the list.

```
<!-- Unordered List -->
<ul>
  <li> Course 1</li>
  <li> Course 2 </li>
  <li> Course 3</li>
</ul>
```

```
<!-- Ordered List -->
<ol>
  <li> Course 1 </li>
  <li> Course 2 </li>
  <li> Course 3 </li>
</ol>
```

# FORMS IN HTML

The **<form>** element is used to create a form in HTML. Forms are used to gather user input.

Some attributes associated with the **<form>** element include:

- The **action** attribute specifies where the form data should be sent when the form is submitted.
- The **target** attribute specifies where to display the form's response.
- The **autocomplete** attribute can have a value of on or off and determines whether the browser should automatically fill in the form.
- The **novalidate** attribute specifies that the form should not be validated.
- The **method** attribute specifies the HTTP method to use when sending form data.
- The **name** attribute specifies the name of the form.
- The **required** attribute specifies that an input element cannot be left blank.
- The **autofocus** attribute gives focus to the input elements when the page loads.
- The **disabled** attribute disables an input element, preventing the user from interacting with it.
- The **placeholder** attribute is used to provide a hint to the user about what information is required for the input element.

## Other input elements that can be used in forms include:

- **<textarea>**: allows users to enter multiple lines of text as input.
- **<select>**: provides a list of options for users to choose from.
- **<option>**: creates a single option within a **<select>** element.
- **<input>**: provides an input field for users to enter data. The **type** attribute specifies the type of data that can be entered.
- **<button>**: creates a button that can be clicked to perform an action.

```
<form action="/Submit_URL/" method="post">
    <label for="FirstName"> First Name: </label>
    <input type="text"
        name="FirstName"
        placeholder="First Name"
        required >

    <label for="LastName"> Last Name: </label>
    <input type="text"
        name="LastName"
        placeholder="Last Name"
        required >
    <label for="add"> Address: </label>
    <textarea name="add"></textarea>
    <label for="age"> Age: </label>
    <select id="age">
        <option value="11-20">11-20</option>
        <option value="21-30">21-30</option>
        <option value="31-40">31-40</option>
        <option value="41-50">41-50</option>
    </select>

    <input type="submit" value="Submit">
</form>
```

# TABLES IN HTML

The **<table>** tag defines a HTML table.

- **<thead>**: defines the header information for each column in the table.
- **<tbody>**: defines the body or content of the table.
- **<tfoot>**: defines the footer information of the table. **<tr>**: represents a row in the table.
- **<td>**: represents a single cell in the table.
- **<th>**: represents the heading for a column of values in the table.

```
<table>
  <thead>
    <tr>
      <th> Name </th>
      <th> CGPA </th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td> Koushik Sadhu </td>
      <td> 9.66 </td>
    </tr>
    <tr>
      <td> Pranay Gupta </td>
      <td> 9.72 </td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td> Nidhi Gupta </td>
      <td> 10 </td>
    </tr>
  </tfoot>
</table>
```

# TAGS IN HTML

The following tags were introduced in HTML5:

- **<header> tag:** defines the header section of a webpage.
- **<footer> tag:** defines the footer section of a webpage.
- **<main> tag:** defines the main content section of a webpage.
- **<article> tag:** defines a standalone section of content, such as an article.
- **<nav> tag:** used to contain navigation links.
- **<meter> tag:** used to measure data within a given range.
- **<progress> tag:** used as a progress bar to indicate the completion of a task.
- **<dialog> tag:** used to create a dialog box.
- **<audio> tag:** used to embed an audio file on a webpage.
- **<video> tag:** used to embed a video on a webpage.
- **<section> tag:** defines a section within a webpage.
- **<aside> tag:** often used for content placed in a sidebar.
- **<time> tag:** used for formatting dates and times.
- **<figure> tag:** used for figures such as charts.
- **<figcaption> tag:** provides a description for a **<figure>**.

```
<header>
    <h1> Welcome to CodeHelp! </h1>
</header>

<nav>
    <ul>
        <li><a href="#">About Us</a></li>
        <li><a href="#">Courses</a></li>
        <li><a href="#">Contact</a></li>
    </ul>
</nav>

<article>
    <h1> About CodeHelp </h1>
    <p> This is all about CodeHelp. </p>
    <aside>
        <p> Book your seat now. </p>
    </aside>
</article>

<progress min="0" max="100" value="10"> </progress>

<footer> Copyright © 2023 CodeHelp. All Rights Reserved. </footer>
```

# CHARACTER AND SYMBOLS

In HTML documents, some symbols may not be directly available on the keyboard. However, there are several ways to include these symbols in a document. These include using the symbol's entity name, decimal value, or hexadecimal value.

- Copyright Symbol: &copy;
- Dollar Symbol: &dollar;
- Ampersand Symbol: &amp;
- Greater than Symbol: &gt;
- Less than Symbol: &lt;

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title> Code Help HTML Cheat Sheet </title>
  </head>
  <body>
    <p> Copyright Symbol: &copy; </p>
    <p> Dollar Symbol: &dollar; </p>
    <p> Ampersand Symbol: &amp;lt; </p>
    <p> Greater than Symbol: &gt; </p>
    <p> Less than Symbol: &lt; </p>
  </body>
</html>
```



# { } CSS CheatSheet



In this Cheatsheet, we will cover the basics of CSS properties. We will provide examples to help you understand how these properties work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.



**CSS (Cascading Style Sheets)** is a stylesheet language used for describing the look and formatting of a document written in HTML (Hypertext Markup Language).

CSS is used to define the layout, font, color, and other visual aspects of a webpage, and it helps to separate the content of a webpage from its presentation. CSS allows you to apply styles to multiple HTML elements at once, and it makes it easy to maintain and update the styling of a webpage.

You can use CSS to specify styles for different devices, such as desktop computers, tablets, and mobile phones, which makes it easier to create responsive designs that look good on any device. To use CSS, you can include a stylesheet in your HTML document using a `<link>` element, or you can use inline styles in your HTML elements using the `style` attribute.

You can also use CSS to style elements in other documents, such as XML or SVG, and you can use CSS in combination with other technologies, such as JavaScript, to create dynamic and interactive webpages.

# FONT PROPERTIES IN CSS

The font has many properties that you can change, such as its face, weight, and style, which allow you to alter the appearance of your text.

- **Font-Family:** Specifies the font family of the text.
- **Font:** A shorthand property for specifying the font-style, font-variant, font-weight, font-size/line-height, and font-family properties all at once.
- **Font-Size:** Specifies the font size of the text.
- **Font-Weight:** Specifies the weight of a font.
- **Font-Style:** Specifies the font style of the text.
- **Font-Variant:** Specifies whether or not the text is displayed in a small-caps font.

```
p {  
    font-family: Times, serif, Arial, Helvetica, sans-serif;  
    font: 15px Helvetica, sans-serif, Arial;  
    font-size: 15px;  
    font-weight: bold;  
    font-style: italic;  
    font-variant: small-caps;  
}
```

# TEXT PROPERTIES IN CSS

CSS has a lot of properties for formatting text.

- **Text-Align:** It specifies the horizontal alignment of text.
- **Text-Decoration:** It specifies the decoration added to text.
- **Letter-Spacing:** It increases or decreases the space between characters in a text.
- **Text-Transform:** It controls the capitalization of text.
- **Word-Spacing:** It increases or decreases the space between words in a text.
- **Text-Indent:** It specifies the indentation of the first line in a text-block.
- **Line-Height:** It sets the line height.
- **Text-Shadow:** It adds shadow to the text.

```
p {  
    text-align: center;  
    text-decoration: underline;  
    letter-spacing: 5px;  
    text-transform: uppercase;  
    word-spacing: 8px;  
    text-indent: 40px;  
    line-height: 40%;  
    text-shadow: 4px 4px #ff0000;  
}
```

# BACKGROUND PROPERTIES IN CSS

The background properties allow you to customize the color, image, position, size, and other aspects of the document's background. As the name implies, these properties affect the background of the document.

- **Background-Image:** It specifies one or more background images for an element.
- **Background-Size:** It specifies the size of the background images.
- **Background-Position:** It specifies the position of a background image.
- **Background-Repeat:** It sets, how a background image will be repeated.
- **Background-Color:** It specifies the background color of an element.
- **Background-Attachment:** It sets whether a background image scrolls with the rest of the page, or is fixed.
- **Background-Origin:** It specifies the origin position of a background image.
- **Background:** A shorthand property for all the background-\* properties.

```
body {  
    background-image: url('codeHelp.png');  
    background-size: auto;  
    background-position: center;  
    background-repeat: no-repeat;  
    background-color: #ffffff;  
    background-attachment: fixed;  
    background-origin: content-box;  
    background: url('codeHelp.png');  
}
```

## BORDERS PROPERTIES IN CSS

The border properties enable you to alter the style, radius, color, and other characteristics of the buttons or other elements within the document.

- **Border-Width:** It sets the width of the four borders.
- **Border-Style:** It sets the style of the four borders.
- **Border-Radius:** A shorthand property for the four border-\*-radius properties.
- **Border-Color:** It sets the color of the four borders.

- **Border:** A shorthand property for border-width, border-style and border-color.
- **Border-Spacing:** It sets the distance between the borders of adjacent cells.

```
div {  
    border-width: 4px;  
    border-style: solid;  
    border-radius: 4px;  
    border-color: #000000;  
    border: 20px dotted coral;  
    border-spacing: 20px;  
}
```

## BOX MODEL PROPERTIES IN CSS

The CSS box model is a structure that encloses every HTML element, and is composed of margins, borders, padding, and the element's content. This model is utilized to design and arrange the layout of web pages.

- **Padding:** A shorthand property for all the padding-\* properties.
- **Visibility:** It specifies whether or not an element is visible.
- **Display:** It specifies how a certain HTML element should be displayed.

- **Height:** It sets the height of an element.
- **Width:** It sets the width of an element.
- **Float:** It specifies whether an element should float to the left, right, or not at all.
- **Clear:** It specifies what should happen with the element that is next to a floating element.
- **Margin:** It sets all the margin properties in one declaration.
- **Overflow:** It specifies what happens if content overflows an element's box.

```
p {  
    padding: 10px 20px 10px 20px;  
    visibility: hidden;  
    display: inline-block;  
    height: auto;  
    width: 100px;  
    float: right;  
    clear: left;  
    margin: 20px 10px 20px 10px;  
    overflow: scroll;  
}
```

# COLORS PROPERTIES IN CSS

The color property can be used to add color to various objects.

- **Color:** It sets the color of text.
- **Outline-Color:** It sets the color of an outline.
- **Caret-Color:** It specifies the color of the cursor (caret) in inputs, textareas, or any element that is editable.
- **Opacity:** It sets the opacity level for an element.

```
{  
    color: rgb(0, 0, 0);  
    outline-color: #000000;  
    caret-color: coral;  
    opacity: 0.8;  
}
```

# LAYOUT PROPERTIES IN CSS

It defines the appearance of the content within a template.

- **Box-Align:** It specifies how an element aligns its contents across its layout in a perpendicular direction.
- **Box-Direction:** It specifies whether a box lays out its contents normally (from the top or left edge), or in reverse (from the bottom or right edge).
- **Box-Flex:** This property specifies how a box grows to fill the box that contains it, in the direction of the containing box's layout.
- **Box-Orient:** It sets whether an element lays out its contents horizontally or vertically.
- **Box-Sizing:** It allows us to include the padding and border in an element's total width and height.
- **Box-Pack:** This property specifies how a box packs its contents in the direction of its layout.
- **Min-Width:** It sets the minimum width of an element.
- **Max-Width:** It sets the maximum width of an element.
- **Min-Height:** It sets the minimum height of an element.
- **Max-Height:** It sets the maximum height of an element.

```
{  
  box-align: start;  
  box-direction: normal;  
  box-flex: normal;  
  box-orient: inline;  
  box-sizing: margin-box;  
  box-pack: justify;  
  min-width: 200px;  
  max-width: 400px;  
  min-height: 100px;  
  max-height: 1000px;  
}
```

## TABLE PROPERTIES IN CSS

Table properties allow you to customize the appearance of tables in a document, including options like border spacing, table layout, and captions.

- **Border-Spacing:** It sets the distance between the borders of adjacent cells.
- **Border-Collapse:** It sets whether table borders should collapse into a single border or be separated.
- **Empty-Cells:** It specifies whether or not to display borders and background on empty cells in a table.
- **Caption-Side:** It specifies the placement of a table caption.

- **Table-Layout:** It defines the algorithm used to layout table cells, rows, and columns.

```
{  
    border-spacing: 4px;  
    border-collapse: separate;  
    empty-cells: show;  
    caption-side: bottom;  
    table-layout: auto;  
}
```

## COLUMNS PROPERTIES IN CSS

These properties are specifically applied to the columns of a table and are used to enhance its appearance.

- **Column-Gap:** It specifies the gap between the columns.
- **Column-Rule-Width:** It specifies the width of the rule between columns.
- **Column-Rule-Color:** It specifies the color of the rule between columns.
- **Column-Rule-Style:** It Specifies the style of the rule between columns.
- **Column-Count:** It specifies the number of columns an element should be divided into.
- **Column-Span:** It specifies how many columns an element should span across.

- **Column-Width:** It specifies the column width.

```
{  
    column-gap: 4px;  
    column-rule-width: medium;  
    column-rule-color: #000000;  
    column-rule-style: dashed;  
    column-count: 20;  
    column-span: all;  
    column-width: 4px;  
}
```

## LIST & MARKER PROPERTIES IN CSS

The list and marker properties can be used to customize the appearance of lists in a document.

- **List-Style-Image:** It specifies an image as the list-item marker.
- **List-Style-Position:** It specifies the position of the list-item markers (bullet points).
- **List-Style-Type:** It specifies the type of list-item marker.
- **Marker-Offset:** It allows you to specify the distance between the marker and the text relating to that marker.

```
{  
    list-style-image: url('codeHelp.png');  
    list-style-position: 10px;  
    list-style-type: square;  
    marker-offset: auto;  
}
```

## ANIMATION PROPERTIES IN CSS

CSS animations enable the creation of animated transitions or the animation of other media elements on a web page.

- **Animation-Name:** It specifies a name for the @keyframes animation.
- **Animation-Delay:** It specifies a delay for the start of an animation.
- **Animation-Duration:** It specifies a delay for the start of an animation.
- **Animation-Timing-Function:** It specifies the speed curve of an animation.
- **Animation-Iteration-Count:** It specifies the number of times an animation should be played.
- **Animation-Fill-Mode:** It specifies a style for the element when the animation is not playing (before it starts, after it ends, or both).

- **Animation-Play-State:** It specifies whether the animation is running or paused.
- **Animation-Direction:** It specifies whether the animation is running or paused.

```
{  
    animation-name: anime;  
    animation-delay: 4ms;  
    animation-duration: 10s;  
    animation-timing-function: ease;  
    animation-iteration-count: 5;  
    animation-fill-mode: both;  
    animation-play-state: running;  
    animation-direction: normal;  
}
```

## TRANSITION PROPERTIES IN CSS

Transitions allow you to specify how an element will change from one state to another.

- **Transition-Property:** It specifies the name of the CSS property the transition effect is for.
- **Transition-Delay:** It specifies when the transition effect will start.
- **Transition-Duration:** It specifies how many seconds or milliseconds a transition effect takes to complete.

- **Transition-Timing-Function:** It specifies the speed curve of the transition effect.

```
{  
    transition-property: none;  
    transition-delay: 4ms;  
    transition-duration: 10s;  
    transition-timing-function: ease-in-out;  
}
```

## CSS FLEXBOX

Flexbox is a CSS layout system that makes it easy to align and distribute items within a container using rows and columns. It allows items to "flex" and adjust their size to fit the available space, making responsive design simpler to implement. Flexbox makes formatting HTML elements more straightforward and efficient.

### PARENT PROPERTIES:

- **Display:** It specifies how a certain HTML element should be displayed.
- **Flex-Direction:** It specifies the direction of the flexible items.
- **Flex-Wrap:** It specifies whether the flexible items should wrap or not.
- **Flex-Flow:** It is a shorthand property for the flex-direction and the flex-wrap properties.

- **Justify-Content:** It specifies the alignment between the items inside a flexible container when the items do not use all available space.
- **Align-Items:** It specifies the alignment for items inside a flexible container.
- **Align-Content:** It specifies the alignment between the lines inside a flexible container when the items do not use all available space.

```
{  
    display: flex;  
    flex-direction: row | row-reverse | column | column-reverse;  
    flex-wrap: nowrap | wrap | wrap-reverse;  
    flex-flow: column wrap;  
    justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly | start | end | left | right ... + safe | unsafe;  
    align-items: stretch | flex-start | flex-end | center | baseline | first baseline | last baseline | start | end | self-start | self-end + ... safe | unsafe;  
    align-content: flex-start | flex-end | center | space-between | space-around | space-evenly | stretch | start | end | baseline | first baseline | last baseline + ... safe | unsafe;  
}
```

## CHILD PROPERTIES:

- **Order:** It sets the order of the flexible item, relative to the rest.
- **Flex-Grow:** It specifies how much the item will grow relative to the rest.
- **Flex-Shrink:** It specifies how the item will shrink relative to the rest.
- **Flex-Basis:** It Specifies the initial length of a flexible item.
- **Align-Self:** It specifies the initial length of a flexible item.

```
{  
    order: 2;                      /* By default it is 0 */  
    flex-grow: 5;                   /* By default it is 0 */  
    flex-shrink: 4;                 /* By default it is 1 */  
    flex-basis: | auto;             /* By default it is auto */  
    align-self: auto | flex-start | flex-end | center | baseline | stretch;  
    flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ];  
}
```

# CSS GRID

The Grid layout is a 2-dimensional system in CSS that allows for the creation of complex and responsive web design layouts with consistent results across different browsers. It makes it easier to build these types of layouts.

## PARENT PROPERTIES:

- **Display:** It specifies how a certain HTML element should be displayed.
- **Grid-Template-Columns:** It specifies the size of the columns, and how many columns in a grid layout.
- **Grid-Template-Rows:** It specifies the size of the columns, and how many columns in a grid layout.
- **Grid-Template:** It is a shorthand property for the grid-template-rows, grid-template-columns and grid-areas properties.
- **Column-Gap:** It specifies the gap between the columns.
- **Row-Gap:** It specifies the gap between the grid rows.
- **Grid-Column-Gap:** It specifies the size of the gap between columns.
- **Grid-Row-Gap:** It specifies the size of the gap between rows.
- **Gap:** It is a shorthand property for the row-gap and the column-gap properties.

- **Grid-Gap:** It is a shorthand property for the grid-row-gap and grid-column-gap properties.
- **Align-Items:** It specifies the alignment for items inside a flexible container.
- **Justify-Content:** It specifies the alignment between the items inside a flexible container when the items do not use all available space.
- **Align-Content:** It specifies the alignment between the lines inside a flexible container when the items do not use all available space.
- **Grid-Auto-Columns:** It specifies a default column size.
- **Grid-Auto-Rows:** It specifies a default row size.
- **Grid-Auto-Flow:** It specifies how auto-placed items are inserted in the grid.

```
{  
  display: grid | inline-grid;  
  grid-template-columns: 10px 10px 10px;  
  grid-template-rows: 5px auto 10px;  
  grid-template: none | <grid-template-rows> / <grid-template-  
  columns>;  
  column-gap: <line-size>;  
  row-gap: <line-size>;  
  grid-column-gap: <line-size>;  
  grid-row-gap: <line-size>;  
  gap: <grid-row-gap> <grid-column-gap>;  
  grid-gap: <grid-row-gap> <grid-column-gap>;  
  align-items: start | end | center | stretch;  
  justify-content: start | end | center | stretch | space-around | space-  
  between | space-evenly;  
  align-content: start | end | center | stretch | space-around | space-  
  between | space-evenly;  
  grid-auto-columns: <track-size>;  
  grid-auto-rows: <track-size>;  
  grid-auto-flow: row | column | row dense | column dense;  
}  
}
```

## CHILD PROPERTIES:

- **Grid-Column-Start:** It specifies where to start the grid item.
- **Grid-Column-End:** It specifies where to end the grid item.
- **Grid-Row-Start:** It specifies where to start the grid item.
- **Grid-Row-End:** It specifies where to end the grid item.
- **Grid-Column:** It is a shorthand property for the grid-column-start and the grid-column-end properties.
- **Grid-Row:** It is a shorthand property for the grid-row-start and the grid-row-end properties.
- **Grid-Area:** It either specifies a name for the grid item, or this property is a shorthand property for the grid-row-start, grid-column-start, grid-row-end, and grid-column-end properties.
- **Align-Self:** It specifies the alignment for selected items inside a flexible container.

```
{  
  grid-column-start: <number> | <name> | span <number> | span <name>  
  | auto;  
  grid-column-end: <number> | <name> | span <number> | span <name> |  
  auto;  
  grid-row-start: <number> | <name> | span <number> | span <name> |  
  auto;  
  grid-row-end: <number> | <name> | span <number> | span <name> |  
  auto;  
  grid-column: <start-line> / <end-line> | <start-line> / span <value>;  
  grid-row: <start-line> / <end-line> | <start-line> / span <value>;  
  grid-area: <name> | <row-start> / <column-start> / <row-end> /  
  <column-end>;  
  align-self: start | end | center | stretch;  
}
```

# Bootstrap Cheat Sheet

If you plan to pick up some coding skills, Bootstrap 4 is a solid choice!

## Why?

It is the gold standard of front-end development:

.active

Bootstrap has built-in classes, meaning you don't have to code most elements from scratch.



It's open-source and has major community support.



Plus, it's super versatile and can be used to create any type of website in no time.

## What is Bootstrap?

Bootstrap 4 is a popular framework for front-end website development.

Primarily, it is a CSS mobile-first design and includes both CSS and JavaScript templates for such things as forms, buttons, navigation, typography, dropdowns, popovers, modals, and carousels, along with other interface elements. But don't misjudge it: Bootstrap also offers plenty of room for customization and you can use it to code any type of website.

You can check out full documentation on the official website.



# Key Bootstrap Components

## Bootstrap.css

This is the basic Bootstrap package that you will need to download. CSS is a style sheet language for static information.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGs03+Hhxv8T/Q5PaXtkKtu6ug5T0eNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
```

## Bootstrap.js

A JavaScript/JQuery library that powers up certain components of Bootstrap such as animation, scrolling, and interactivity.

```
<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849bleE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIYzFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUDj003uMBJnjuUD4IH7YwaYd1iqfktj0Uod8GCExl30g8ifwB6" crossorigin="anonymous"></script>
```

## Glyphicons

Glyphs are elemental symbols with typography, such as the English Pound symbol (£). Bootstrap has a huge list of embedded glyph icons that are available for free.

# Bootstrap source code elements

The Bootstrap source code download includes the precompiled CSS, JavaScript, and font assets, along with source Less, JavaScript, and documentation.

**less/** - a preprocessor style sheet for CSS that eliminate repetitive coding tasks

**sass/** - a newer version of the preprocessor that is more popular

**js/** - simply refers to the source code JavaScript, which allows Bootstrap components to work

**fonts/** - these are icon fonts that come with the download

**dist/** - a folder that contains precompiled files for drop-in use in website development

Note: Bootstrap also requires jQuery installation for Bootstrap's JavaScript plugins. jQuery is a feature-rich component of the JavaScript library, and it whittles down lots of JavaScript code and wraps them into actions you can accomplish with a single line.

```
<script src="https://code.jquery.com/jquery.js"></script>
```

To install plug-ins:

```
<script src="js/bootstrap.min.js"></script>
```

# Bootstrap Screen Sizes

Bootstrap 4 is a mobile-first responsive framework. But still you need to provide screen size instructions when you are creating page layout grids. Below are the standard screen sizes for reference:



## Min:

```
@media (min-width: @screen-sm-min) // >= 768px (small tablet)
@media (min-width: @screen-md-min) // >= 992px (medium laptop)
@media (min-width: @screen-lg-min) // >= 1200px (large desktop)
```

## Max:

```
@media (max-width: @screen-xs-max) { // < 768px (xsmall phone)
@media (max-width: @screen-sm-max) { // < 992px (small tablet)
@media (max-width: @screen-md-max) { // < 1200px (medium laptop)
```

# Bootstrap Key Components

## Alerts

Alerts are messages to users when something is wrong. For example, if a user types in an incorrect email address or credit card number, they'll receive an error alert message, prompting them to make corrections.

### .alert-primary

You can create colorful alerts for any texts. Primary alert (more important message) is in light blue color.

```
<div class="alert alert-primary" role="alert">Primary alert</div>
```

### .alert-secondary

Add a secondary alert (less important message) in light gray color.

```
<div class="alert alert-secondary" role="alert">I'm a secondary alert</div>
```

### .alert-success

This will alert a user that their action has been successful

```
<div class="alert alert-success" role="alert">Success alert!</div>
```

### .alert-warning

This will send a message of an upcoming action.

```
<div class="alert alert-warning" role="alert">Warning! Warning!</div>
```

### .alert-danger

A danger alert is similar to a warning alert, but for more negative actions (e.g., getting locked out after too many password fails)

```
<div class="alert alert-danger" role="alert"> You are in grave danger, my friend!</div>
```

### .alert-link

So you want to add another message and a link to that message in the original alert? You can embed that message and in the same color.

```
<div class="alert alert-success"> <strong>Success!</strong> You should <a href="#" class="alert-link">read this cool message</a>. </div>
```

### .alert-dismissible

Provides an option to close the alert after reading it.

```
<div class="alert alert-success alert-dismissible"> <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a><strong>Success! You did well</strong> </div>
```

### .alert-heading

Add a quick header to your alert. (e.g., "shipping address" or "billing address"). It could relate to an entire page or just a piece of content within that page.

```
<div class="alert alert-info">
<h4 class="alert-heading"><i class="fa fa-info"></i>
```

### .alert-light and .alert-dark

Changes the alert style to an in either a light or dark gray color.

```
<div class="alert alert-light" role="alert"> I'm the light alert </div>
<div class="alert alert-dark" role="alert">And I'm the dark (side) alert!</div>
```

# Badge

Use badges to display extra information next to a category, button, or other element. You can create and style them with other context elements (e.g., .badge-warning). Also, badges will scale to match the size of the parent element (e.g. headings). Lastly, you can use badges as a part of buttons or links to provide counters.

## Example: Headings

Suppose you have a number of headings and you are adding a badge. That badge will adjust in size to match the heading.

```
<h1>Example heading <span class="badge badge-secondary">New</span></h1>
<h2>Example heading <span class="badge badge-secondary">New</span></h2>
```

### .badge-pill

Use this command to modify the shape of your badges, making them more rounded.

```
<span class="badge badge-pill badge-light">Light</span>
```

### .badge-primary + .badge-secondary

You may want to add a badge to primary (more important) and secondary (less important) messages.

```
<span class="badge badge-primary">Primary</span>
<span class="badge badge-secondary">Secondary</span>
```

### .badge-transparent

Suppose you want to make a button transparent to make it stand out from the rest. With Bootcamp 4, you do not have to use in-line styling. Instead, use the following command:

```
<button class="btn bg-transparent"></button>
```

### .badge-warning, -success, -info, or -danger

Adds a badge to a warning, success, info or danger alert.

```
<span class="badge badge-warning">Warning</span>
<span class="badge badge-success">Success</span>
<span class="badge badge-danger">Danger</span>
<span class="badge badge-info">Info</span>
```

## Create Actionable Badges with Hover and Focus States

When users move their cursors to a button but have not yet activated it, you can provide hover and focus elements to encourage action.

To do this, use the contextual .badge-\* classes on an  [element. Here are a few examples:](#)

```
<a href="#" class="badge badge-primary">Primary</a>
<a href="#" class="badge badge-secondary">Secondary</a>
<a href="#" class="badge badge-success">Success</a>
<a href="#" class="badge badge-danger">Danger</a>
<a href="#" class="badge badge-warning">Warning</a>
```

## Breadcrumbs

Breadcrumbs are navigational components that will help users move from page to page without getting lost and give them the way to pedal back to a previous page.

```
<nav aria-label="breadcrumb">
  <ol class="breadcrumb">
    <li class="breadcrumb-item active" aria-current="page">Home</li>
  </ol>
</nav>

<nav aria-label="breadcrumb">
  <ol class="breadcrumb">
    <li class="breadcrumb-item"><a href="#">Home</a></li>
    <li class="breadcrumb-item active" aria-current="page">Library</li>
  </ol>
</nav>

<nav aria-label="breadcrumb">
  <ol class="breadcrumb">
    <li class="breadcrumb-item"><a href="#">Home</a></li>
    <li class="breadcrumb-item active" aria-current="page">Library</li>
    <li class="breadcrumb-item active" aria-current="page">Data</li>
  </ol>
</nav>
```

## buttons

As the name hints,.button command lets you create and style ... a button.

### .button-primary + .button-secondary

A primary button is commonly used for a user action; a secondary button may then be used to close out.

```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
```

### .btn-light, -dark .btn-primary, -secondary, -transparent, -white, -warning, -success, -info, -danger

Design your buttons using the standard predefined styling options:

```
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-dark">Dark</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="button-transparent">Transparent</button>
<button type="button" class="button-white">White</button>
```

### .btn-outline

Choose a button outline following these snippet samples:

```
<button type="button" class="btn btn-outline-primary">Primary</button>
```

### .btn-lg + .btn-sm

Change the size of your buttons.

```
<button type="button" class="btn btn-primary btn-lg">I'm the large button</button>
<button type="button" class="btn btn-primary btn-sm">I'm a small button</button>
```

### .btn-block

Group your buttons into a block. All grouped buttons will span the full width of a parent.

```
<button type="button" class="btn btn-primary btn-lg btn-block">Block level button</button>
```

### .active

By default, all buttons will be displayed as pressed - with a dark border, darker background and inset shadow - when active.

You don't need to add a class to `<button>`s as they use a pseudo-class.

But, if for some reason, you need to force that same active look, use the following code snippet:

```
<a href="#" class="btn btn-primary btn-lg active" role="button" aria-pressed="true">Primary link</a>
```

Note: You can also add the disabled Boolean attribute to any button to make it look inactive.

```
<button type="button" class="btn btn-lg btn-primary" disabled>Disabled button</button>
```

## Button Group

Use this element to make a group of similarly-sized buttons without coding each separately.

### .btn-group

```
<div class="btn-group" role="group" aria-label="Basic example">
  <button type="button" class="btn btn-secondary">Left</button>
  <button type="button" class="btn btn-secondary">Middle</button>
  <button type="button" class="btn btn-secondary">Right</button>
</div>
```

### .btn-group-vertical

Style a vertical group of buttons:

```
<div class="btn-group-vertical" role="group" aria-label="Basic example">
  <button type="button" class="btn btn-secondary">Left</button>
  <button type="button" class="btn btn-secondary">Middle</button>
  <button type="button" class="btn btn-secondary">Right</button>
</div>
```

### .btn-group (Nested)

You can also create nested buttons with drop down menus.

```
<div class="btn-group" role="group" aria-label="Button group with nested dropdown">
  <button type="button" class="btn btn-secondary">1</button>
  <button type="button" class="btn btn-secondary">2</button>

  <div class="btn-group" role="group">
    <button id="btnGroupDrop1" type="button"
      class="btn btn-secondary dropdown-toggle"
      data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
      Dropdown
    </button>
    <div class="dropdown-menu" aria-labelledby="btnGroupDrop1">
      <a class="dropdown-item" href="#!>Dropdown link</a>
      <a class="dropdown-item" href="#!>Dropdown link</a>
    </div>
  </div>
</div>
```

### .btn-toolbar

Arrange button groups into a toolbar to make more complex components. You can apply different utility classes for additional styling.

```
<div class="btn-toolbar" role="toolbar" aria-label="Toolbar with button groups">
  <div class="btn-group" role="group" aria-label="First group">
    <button type="button" class="btn btn-secondary">1</button>
    <button type="button" class="btn btn-secondary">2</button>
    <button type="button" class="btn btn-secondary">3</button>
  </div>
  <div class="btn-group" role="group" aria-label="Second group">
    <button type="button" class="btn btn-secondary">5</button>
    <button type="button" class="btn btn-secondary">6</button>
  </div>
  <div class="btn-group" role="group" aria-label="Third group">
    <button type="button" class="btn btn-secondary">8</button>
  </div>
</div>
```

### .btn-group-toggle

Install Bootstrap Toggle plugin to modify checkboxes into toggles. You can then add `data-toggle="buttons"` to a button group with modified buttons to enable their toggling behavior via JavaScript. Afterwards, use `.btn-group-toggle` to style different inputs within your buttons.

```
<div class="btn-group-toggle" data-toggle="buttons">
  <label class="btn btn-secondary active">
    <input type="checkbox" checked autocomplete="off"> Checked
  </label>
</div>
```

# Cards

Cards are flexible containers with options for headers/footers, colors and display options, and more. They replaced several earlier components (panels, wells, and thumbnails) from Bootstrap 3.

### .card-body

The main element of the card. Use it to add a padded section within your card.

```
<div class="card">
  <div class="card-body">
    Your awesome text
  </div>
</div>
```

### .card-title

Code a title for your card. Add this to a `<h*` tag.

```
<h4 class="card-title">Big title</h4>
```

### .card-subtitle

You can also add subtitles to every card for some extra fanciness.

```
<h6 class="card-subtitle mb-2 text-muted">Fancy card subtitle</h6>
```

### .card-link

Embed a link inside your card. Add this class to an `<a>` tag.

```
<a href="#" class="card-link">Link to something fun</a>
```

### .card-text

Add some words to your card. As many or as few as you want.

```
<p class="card-text">  
Roses are red, violets are blue,  
I'm learning Bootstrap,  
What about you?  
</p>
```

### .card-img-top

You can also enclose an image to your card. This snippet will add one atop of it.

```
<div class="card">  
    
  <div class="card-body"> What an epic image!</div>  
</div>
```

### .card-img-bottom

Or you can have the image displayed as a bottom of the card. Your call.

```
<div class="card">  
  <div class="card-body"> Look at that pic!</div>  
    
</div>
```

### .card-img-overlay

Use an image as a background and overlay all the texts.

```
<div class="card">  
    
  <div class="card-img-overlay">  
    <p class="card-text">I'm text that has a background image!</p>  
  </div>  
</div>
```

### .card-header

Place a custom header at the top of your card. It will be displayed above all the titles and subtitles.

```
<div class="card">  
  <div class="card-header">  
    Some big header!  
  </div>
```

### .card-footer

Also, you can code a footer for your card to communicate some extra info. It will go right after the .card-body.

```
<div class="card">  
  <div class="card-body">  
    <p class="card-text">Some more card content</p>  
  </div>  
  <div class="card-footer">  
    Updated 2 days ago  
  </div>  
</div>
```

## .card-group

Play around with card layout and build a group of cards. A group will act as a single, attached element, with every card having the same width and height. You can also apply `display: flex;` to improve sizing.

Note: Group card layouts are not responsive!

```
<div class="card-group">
  <div class="card">
    <div class="card-body">
      <h4 class="card-title">Card number one</h4>
      <p class="card-text">I'm the first card in the group, displaying some cool
        info.</p>
    </div>
  </div>
  <div class="card">
    <div class="card-body">
      <h4 class="card-title">Card number two</h4>
      <p class="card-text">I'm the middle card in the group and probably I offer the
        best deal</p>
    </div>
  </div>
  <div class="card">
    <div class="card-body">
      <h4 class="card-title">Card three</h4>
      <p class="card-text">I'm the third card trying to be as cool as the first
        one.</p>
    </div>
  </div>
</div>
```

## .card-columns

You can organize your cards into Masonry-like columns. This allows you to build some creative patterns using only CSS.

NB: If your cards are breaking across columns, set them to `display: inline-block;`

```
<div class="card-columns">
  <div class="card">
    <div class="card-body">
      <!-- Card content -->
    </div>
  </div>
  <div class="card p-3">
    <!-- Card content -->
  </div>
  <div class="card">
    <div class="card-body">
      <!-- Card content -->
    </div>
  </div>
  <div class="card bg-primary p-3 text-center">
    <!-- Card content -->
  </div>
</div>
```

### .card-deck

Assemble a set of non-attached cards with equal height and width.

```
<div class="card-deck">
  <div class="card">
    <div class="card-body">
      <h4 class="card-title">Some title</h4>
      <p class="card-text">Your texts </p>
    </div>
  </div>
  <div class="card">
    <div class="card-body">
      <h4 class="card-title">Another card</h4>
      <p class="card-text">Even more texts that someone will need to write.</p>
    </div>
  </div>
  <div class="card">
    <div class="card-body">
      <h4 class="card-title">I'm a card too!</h4>
      <p class="card-text">Some words to add and arrange</p>
    </div>
  </div>
</div>
```

## Carousel

Set up a slideshow to cycle through a series of slides, text, or images. Built with CSS 3D and some JS. You can add images, text or custom markup, as well as add or remove previous/next controls.

### Carousel Slide – creating a single slide

```
<div id="carouselExampleIndicators" class="carousel slide" data-ride="carousel">
  <div class="carousel-inner" role="listbox">
    <div class="carousel-item active">
      
    </div>
    <div class="carousel-item">
      
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleIndicators" role="button"
data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleIndicators" role="button"
data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

### .carousel-fade

Add this cool fade out effect for a slide before the next one.

```
<div id="carouselFadeExampleIndicators" class="carousel slide carousel-fade" data-ride="carousel">
  <div class="carousel-inner" role="listbox">
    <div class="carousel-item active">
      
    </div>
    <div class="carousel-item">
      
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselFadeExampleIndicators" role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselFadeExampleIndicators" role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

### .carousel-indicators

Add control and support for next/previous and indicators such as slide number.

```
<div id="carouselExampleIndicators" class="carousel slide" data-ride="carousel">
  <ol class="carousel-indicators">
    <li data-target="#carouselExampleIndicators" data-slide-to="0" class="active"></li>
    <li data-target="#carouselExampleIndicators" data-slide-to="1"></li>
  </ol>
  <div class="carousel-inner" role="listbox">
    <div class="carousel-item active">
      
    </div>
    <div class="carousel-item">
      
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleIndicators" role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleIndicators" role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

### .carousel-caption

Add a funky caption to each or several slides.

```
<div class="bd-example">
  <div id="carouselExampleCaptions" class="carousel slide" data-ride="carousel">
    <ol class="carousel-indicators">
      <li data-target="#carouselExampleCaptions" data-slide-to="0" class="active"></li>
    </ol>
```

```

<li data-target="#carouselExampleCaptions" data-slide-to="1"></li>
</ol>
<div class="carousel-inner" role="listbox">
  <div class="carousel-item active">
    
    <div class="carousel-caption d-none d-md-block">
      <h3>Cool Slide Label</h3>
      <p>I'm so proud of myself for coding this slide.</p>
    </div>
  </div>
  <div class="carousel-item">
    
    <div class="carousel-caption d-none d-md-block">
      <h3>Even Cooler label</h3>
      <p>Yep, totally rocking this whole coding thing.</p>
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleCaptions" role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleCaptions" role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
</div>

```

## Collapse & Accordion

Collapse is a JavaScript plugin you can use to hide content under the “collapse” menu. The collapsed element will animate its height from 0 to its normal value when triggered and vice versa.

### .collapse

Hide your content.

```
<div class="collapse" id="collapseExample">
  <div class="card card-body">
    Anim pariatur cliche reprehenderit, enim eiusmod high life accusamus terry richardson ad squid. Nihil anim keffiyeh helvetica, craft beer labore wes anderson cred nesciunt sapiente ea proident.
  </div>
</div>
```

### .collapse show

Display the collapsed content.

```
<div id="demo" class="collapse show"> Some text you've decided to hide </div>
```

### .collapse.options

Activate content as collapsible element; accepts optional objects

```
$( '#myCollapsible' ).collapse({
  toggle: false
})
```

### .accordion

An extension of collapsible behavior to cards. You can use this feature to create an accordion.

Note: You must use .accordion as a wrapper

Accordion Snippet Example with 2 group items

```
<div id="accordion" role="tablist">
  <div class="card">
    <div class="card-header" role="tab" id="headingOne">
      <h5 class="mb-0">
        <a data-toggle="collapse" href="#collapseOne" aria-expanded="true" aria-controls="collapseOne">
          Collapsible Group Item #1
        </a>
      </h5>
    </div>

    <div id="collapseOne" class="collapse show" role="tabpanel" aria-labelledby="headingOne">
      <div class="card-body">
        Here's the first thing I want to hide in this card.
      </div>
    </div>
  </div>
  <div class="card">
    <div class="card-header" role="tab" id="headingTwo">
      <h5 class="mb-0">
        <a class="collapsed" data-toggle="collapse" href="#collapseTwo" aria-expanded="false" aria-controls="collapseTwo">
          Collapsible Group Item #2
        </a>
      </h5>
    </div>
    <div id="collapseTwo" class="collapse" role="tabpanel" aria-labelledby="headingTwo">
      <div class="card-body">
        And here's some other text I'm styling with accordion.
      </div>
    </div>
  </div>
</div>
```

## Custom Forms

Bootstrap 4 has several customized form elements that replace the browser defaults.

### .custom-checkbox

As the name implies, you can build a custom checkbox for your form.

```
<div class="custom-control custom-checkbox">
  <input type="checkbox" class="custom-control-input" id="customCheck1">
  <label class="custom-control-label" for="customCheck1">I have a cool custom
checkbox</label>
</div>
```

### .custom-radio

...and you can style a custom radio too!

```
<div class="custom-control custom-radio">
  <input type="radio" id="customRadio1" name="customRadio" class="custom-control-input">
  <label class="custom-control-label" for="customRadio1">Give this custom radio a
  toggle</label>
</div>
```

### .custom-switch

Lastly, you can also create a stylish custom switch.

```
<div class="custom-control custom-switch">
  <input type="checkbox" class="custom-control-input" id="customSwitch1">
  <label class="custom-control-label" for="customSwitch1">Toggle this switch
element</label>
</div>
```

### .custom-select

Use this command to add a custom select menu.

```
<select class="custom-select">
  <option selected>Open this select menu</option>
  <option value="1">One</option>
  <option value="2">Two</option>
  <option value="3">Three</option>
</select>
```

### .custom-file

Customize user file upload. To do so, add .custom-file class around the input with type="file". Then add the .custom-file-input to it.

```
<div class="custom-file">
  <input type="file" class="custom-file-input" id="customFile">
  <label class="custom-file-label" for="customFile">Choose file</label>
</div>
```

### .custom-range

Design a custom range menu.

```
<label for="customRange1">Basic range</label>
<input type="range" class="custom-range" id="customRange1">
<label class="mt-3" for="customRange3">Range with steps</label>
<input type="range" class="custom-range" min="0" max="5" step="0.5"
id="customRange3">
```

# Dropdowns

Use this plugin to create contextual overlays for displaying lists of user links. It's a handy option for creating menus. Dropdowns are built through Popper.js, part of a third-party library. Thus, Make sure you include popper.min.js before Bootstrap's own JavaScript. Or you can just use bootstrap.bundle.min.js/ bootstrap.bundle.js. Both contain Popper.js. Quick styling tip: all dropdowns in Bootstrap are toggled by clicking, not hovering.

## .dropdown

Add a simple dropdown menu with buttons.

```
<div class="dropdown">
  <button class="btn btn-secondary dropdown-toggle"
    type="button" id="dropdownMenu1" data-toggle="dropdown"
    aria-haspopup="true" aria-expanded="false">
    Dropdown
  </button>
  <div class="dropdown-menu" aria-labelledby="dropdownMenu1">
    <a class="dropdown-item" href="#!">Action</a>
    <a class="dropdown-item" href="#!">Another action</a>
  </div>
</div>
```

## .dropdown-toggle-split

Create split button dropdowns with proper spacing around the dropdown caret.

```
<div class="btn-group">
  <button type="button" class="btn btn-secondary">Dropdown</button>
  <button type="button" class="btn btn-secondary dropdown-toggle dropdown-toggle-split" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    <span class="sr-only">Toggle Dropdown</span>
  </button>
  <div class="dropdown-menu">
    <a class="dropdown-item" href="#!">Action</a>
    <a class="dropdown-item" href="#!">Another action</a>
  </div>
</div>
```

## .dropup

Did you know that you can style a menu coming up rather than down? Now you do!

```
<!-- Default dropup button -->
<div class="btn-group dropup">
  <button type="button" class="btn btn-secondary dropdown-toggle" data-
  toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    Dropup
  </button>
  <div class="dropdown-menu">
    <!-- Dropdown menu links -->
  </div>
</div>
```

## .dropright

Provide the menu to the right of the button.

```
<!-- Default dropright button -->
<div class="btn-group dropright">
  <button type="button" class="btn btn-secondary dropdown-toggle" data-
  toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    Dropright
  </button>
  <div class="dropdown-menu">
    <!-- Dropdown menu links -->
  </div>
</div>
```

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

### .dropleft

...And you can also display the menu on the left.

```
<!-- Default dropleft button -->
<div class="btn-group dropleft">
  <button type="button" class="btn btn-secondary dropdown-toggle" data-
  toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    Dropleft
  </button>
  <div class="dropdown-menu">
    <!-- Dropdown menu links -->
  </div>
</div>
```

### .dropdown-item-text

Add non-interactive dropdown items to your menu.

```
<div class="dropdown-menu">
  <span class="dropdown-item-text">Plain text</span>
  <a class="dropdown-item" href="#">Clickable action</a>
  <a class="dropdown-item" href="#">Another action</a>
  <a class="dropdown-item" href="#">Whatever else you need</a>
</div>
```

### .dropdown-item disabled

You can also choose to disable any menu item(s).

```
<div class="dropdown-menu">
  <a class="dropdown-item" href="#">Active link</a>
  <a class="dropdown-item disabled" href="#">Disabled link</a>
  <a class="dropdown-item" href="#">One more link</a>
</div>
```

### .dropdown-divider

Add a simple divider between menu elements to draw extra attention.

```
<div class="dropdown-divider"></div>
```

### .dropdown-menu-right

Align the entire menu to the right

```
<div class="btn-group">
  <button type="button" class="btn btn-secondary dropdown-toggle" data-
  toggle="dropdown" aria-haspopup="true" aria-expanded="false">
```

This dropdown's menu is right-aligned

```
</button>
<div class="dropdown-menu dropdown-menu-right">
  <button class="dropdown-item" type="button">Action</button>
  <button class="dropdown-item" type="button">Another action</button>
  <button class="dropdown-item" type="button">Something else here</button>
</div>
</div>
```

# Forms

You can easily build attractive forms and code custom styles, layouts and additional elements. In Bootstrap 4, forms also received some new input controls such as number election, email verification and others, along with a bunch of new responsive classes.

## Example of .form-group

```
<form>
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp" placeholder="Provide email">
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Your password</label>
    <input type="password" class="form-control" id="exampleInputPassword1" placeholder="Password">
  </div>
  <div class="form-group form-check">
    <input type="checkbox" class="form-check-input" id="exampleCheck1">
    <label class="form-check-label" for="exampleCheck1">Remember me</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

## .form-control

Use this class to style all textual form controls such as user input, titles, etc.

```
<form>
  <div class="form-group">
    <label for="exampleFormControlInput1">Email address</label>
    <input type="email" class="form-control" id="exampleFormControlInput1" placeholder="name@example.com">
  </div>
```

## .form-control-file

Add this class whenever you need to provide the user with an option to upload a file to the form.

```
<input type="file" class="form-control-file" id="exampleFormControlFile1">
```

## .form-control-lg and .form-control-sm.

Create a visual hierarchy within your form by adding .form-control-lg to make bigger input areas and .form-control-sm to make smaller ones.

Email	col-form-label-sm
Email	col-form-label
Email	col-form-label-lg

```
<input class="form-control form-control-lg" type="text" placeholder=".form-control-lg">
<input class="form-control form-control-sm" type="text" placeholder=".form-control-sm">
```

### .form-control-plaintext

Use this class to correctly display `<input type="text" readonly>` elements in your form. It will replace the default form field styling with plain text, while keeping the correct margin and padding.

```
<form>
  <div class="form-group row">
    <label for="staticEmail" class="col-sm-2 col-form-label">Email</label>
    <div class="col-sm-10">
      <input type="text" readonly class="form-control-plaintext" id="staticEmail" value="email@example.com">
    </div>
  </div>
  <div class="form-group row">
    <label for="inputPassword" class="col-sm-2 col-form-label">Pass</label>
    <div class="col-sm-10">
      <input type="password" class="form-control" id="inputPassword" placeholder="Password">
    </div>
  </div>
</form>
```

### .form-control-range

Set horizontally scrollable range inputs for your form.

```
<form>
  <div class="form-group">
    <label for="formControlRange">Some range input</label>
    <input type="range" class="form-control-range" id="formControlRange">
  </div>
</form>
```

### .form-check

Add checkboxes to your form.

```
<div class="form-check">
  <input class="form-check-input" type="checkbox" value="" id="defaultCheck1">
  <label class="form-check-label" for="defaultCheck1">
    My first checkbox
  </label>
</div>
```

Note: You can also add radio buttons instead of checkboxes using `form-check-input` `type="radio"`.

### .form-check-inline

Create a horizontal list of checkboxes.

```
<div class="form-check form-check-inline">
  <input class="form-check-input" type="checkbox" id="inlineCheckbox1" value="option1">
  <label class="form-check-label" for="inlineCheckbox1">1</label>
</div>
<div class="form-check form-check-inline">
  <input class="form-check-input" type="checkbox" id="inlineCheckbox2" value="option2">
  <label class="form-check-label" for="inlineCheckbox2">2</label>
</div>
```

### .readonly boolean attribute

Specify that certain form input is read only. This will prevent modification of the input's value.

```
<input class="form-control" type="text" placeholder="Readonly input here..." readonly>
```

# Input Group

Input group element lets you create more interactive and attractive form controls. Use it to add texts, icons or buttons on both sides of the input field.

## .input-group - Basic Example

```
<div class="input-group">
  <div class="input-group-prepend">
    <span class="input-group-text">First and last name</span>
  </div>
  <input type="text" aria-label="First name" class="form-control">
  <input type="text" aria-label="Last name" class="form-control">
</div>
```

## .input-group-prepend

Provide additional texts in front of the input.

```
<form>
  <div class="input-group mb-3">
    <div class="input-group-prepend">
      <span class="input-group-text">@</span>
    </div>
    <input type="text" class="form-control" placeholder="Name">
  </div>
```

## .input-group-append

...Or list them behind the input.

```
<div class="input-group mb-3">
  <input type="text" class="form-control" placeholder="Password">
  <div class="input-group-append">
    <span class="input-group-text">@example.com</span>
  </div>
</div>
</form>
```

## .input-group-text

Use this class to style specified texts.

```
<div class="input-group mb-3">
  <input type="text" class="form-control" placeholder="1000" aria-label="Amount
  (rounded to the nearest dollar)" aria-describedby="basic-addon">
  <div class="input-group-append">
    <span class="input-group-text" id="basic-addon">.00</span>
  </div>
</div>
```

## Jumbotron

A flexible component that will help you create big boxes to command more attention to featured content or message. In Bootstrap, jumbotron looks like a grey box with rounded corners that automatically enlarges all the font sizes and texts inside of it.

You can add any HTML and other Bootstrap classes inside a jumbotron.

```
.jumbotron
<div class="jumbotron">
  <h1 class="display-3">Hey, awesome you!</h1>
  <p class="lead">This is a simple hero unit, showing that anyone can be a real
hero!.</p>
  <hr class="my-2">
  <p>Use utility classes for typography and spacing</p>
  <p class="lead">
    <a class="btn btn-primary btn-lg" href="#" role="button">Ready for action</a>
  </p>
</div>
```

### **.jumbotron-fluid**

Slightly changes the look of jumbotron and makes it full-page wide without rounded corners.

```
<div class="jumbotron jumbotron-fluid">
  <div class="container">
    <h1 class="display-3">Fluid jumbotron</h1>
    <p class="lead">I'm taking up the entire horizontal space on the page.</p>
  </div>
</div>
```

## List Group

Use list groups to display series of content, with lots of options for styling and layouts.

### **.list-group - Example**

Create a basic list group with several items.

```
<ul class="list-group">
  <li class="list-group-item">Milk</li>
  <li class="list-group-item">Tea</li>
  <li class="list-group-item">Toffees</li>
</ul>
```

### **.list-group-item active**

Add .active class to highlight the current active selection in the list.

```
<ul class="list-group">
  <li class="list-group-item active">I want this</li>
  <li class="list-group-item">Not this</li>
  <li class="list-group-item">Or this</li>
</ul>
```

### **.list-group-item disabled**

Show that one of the list items is not available/disabled. Some active elements (e.g. links) will require custom JavaScript on top of .disabled to become fully inactive.

```
<ul class="list-group">
  <li class="list-group-item disabled">This product is out of stock</li>
  <li class="list-group-item">But we have this!</li>
  <li class="list-group-item">And also this cool thing</li>
</ul>
```

### .list-group-item-action

Add more interactivity to your list by adding styling effects (disabled, hover, active, etc.) to individual items.

```
<div class="list-group">
  <a href="#" class="list-group-item list-group-item-action active">
    Our prime choice for you!
  </a>
  <a href="#" class="list-group-item list-group-item-action">Some good offer</a>
  <a href="#" class="list-group-item list-group-item-action">Also an option</a>
  <a href="#" class="list-group-item list-group-item-action">Something else on the
list</a>
  <a href="#" class="list-group-item list-group-item-action disabled" tabindex="-1"
aria-disabled="true">And this one's not available</a>
</div>
```

### .list-group-flush

Change the look of our list by removing borders and rounded corners. All the items will be placed edge-to-edge.

```
<ul class="list-group list-group-flush">
  <li class="list-group-item">Chai Latte</li>
  <li class="list-group-item">Matcha Latte</li>
  <li class="list-group-item">Earl Grey Tea</li>
  <li class="list-group-item">Vanilla Rooibos</li>
  <li class="list-group-item">Mate</li>
</ul>
```

### .list-group-horizontal

You can also set your list up horizontally rather than vertically. You can also code the list group to become horizontal starting at a certain breakpoint's min-width using .list-group-horizontal-{sm|md|lg|x|}.

NB: You can't use horizontal list groups with flush list groups at the same time.

```
<ul class="list-group list-group-horizontal">
  <li class="list-group-item">Trains</li>
  <li class="list-group-item">Planes</li>
  <li class="list-group-item">Rockets</li>
</ul>
```

### .list-group-item

(light, dark primary, secondary, transparent, white, warning, success, info, danger)

Apply standard styles to individual list items.

```
<ul class="list-group">
  <li class="list-group-item">All the colors that I have </li>
  <li class="list-group-item list-group-item-primary">Light blue</li>
  <li class="list-group-item list-group-item-secondary">Light gray</li>
  <li class="list-group-item list-group-item-success">Green</li>
  <li class="list-group-item list-group-item-danger">Red</li>
  <li class="list-group-item list-group-item-warning">Yellow</li>
  <li class="list-group-item list-group-item-info">Teal</li>
  <li class="list-group-item list-group-item-light">White</li>
  <li class="list-group-item list-group-item-dark">Gray</li>
</ul>
```

# Media Object

Bootstrap 4 lets you build complex, repetitive components featuring texts and some media. Media objects are a cool tool to build tweet-like elements and featured boxes. Also, they are ridiculously easy to use as they require just two classes.

## .media

Use the .media wrapping and .media-body around the content to create a single media object. Here is a sample for a heading.

```
<div class="media">
  
  <div class="media-body">
    <h5 class="mt-0">Your heading</h5>
    A key message you want to share with the world!
  </div>
</div>
```

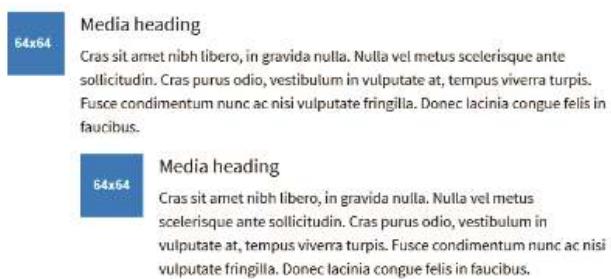
## .media-body

A class specifying what would be inside of your object. You can code different alignments for your content. The default is top, but you can align in the middle or end.

```
<div class="media">
  
  <div class="media-body">
    <h5 class="mt-0">Top-aligned media</h5>
    <p>Read the texts describing the picture above</p>
  </div>
```

## Nested Media

You also have the option to include more than one media object. Objects are nested by beginning at left margin and tabbing each new object in.



Here is a snippet for nesting

```
<div class="media">
  
  <div class="media-body">
    <h5 class="mt-0">Media heading</h5>
```

Here goes some text.

```
<div class="media mt-3">
  <a class="d-flex pr-3" href="#">
    
  </a>
  <div class="media-body">
    <h5 class="mt-0">Another Media heading</h5>
```

Some more texts explaining what's going on here.

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

# Nav

.nav is a base class that helps you build all sorts of navigation components, even with style overrides. You have lots of options for customization.

## .nav example

```
<ul class="nav">
  <li class="nav-item">
    <a class="nav-link disabled" href="#">You can't click this disabled link</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Click this instead</a>
  </li>
</ul>
```

## .nav-items

A class to specify a new item in the navigation menu.

```
</li>
<li class="nav-item">
  <a class="nav-link" href="#">Some item</a>
</li>
```

## .nav justify-content-center

Align your nav horizontally in the center.

```
<ul class="nav justify-content-center">
  <li class="nav-item">
    <a class="nav-link active" href="#">Active</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
</ul>
```

## .nav justify-content-end

Or justify your content to the right.

```
<ul class="nav justify-content-end">
  <li class="nav-item">
    <a class="nav-link active" href="#">Active</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
</ul>
```

## .nav-tabs

Add some cool tabs to your navigation menu.



Note: You'll need tab JavaScript plugin installed.

```
<ul class="nav nav-tabs">
  <li class="nav-item">
    <a class="nav-link active" href="#!>Active</a>
  </li>
  <li class="nav-item">
```

```
<a class="nav-link" href="#!">Link</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="#!">Link</a>
</li>
</ul>
```

### .nav-pills

Alternatively, you can style the menu components as pills.



```
<ul class="nav nav-pills">
<li class="nav-item">
  <a class="nav-link active" href="#!">Active</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="#!">Link</a>
</li>
<li class="nav-item">
  <a class="nav-link disabled" href="#!">Disabled</a>
</li>
</ul>
```

### .Nav-justified

Equalize the widths of all tabs/pills by adding .nav-justified to .nav, .nav-tabs, or .nav, .nav-pills.

Justified Nav Elements Example

```
<ul class="nav nav-pills nav-justified">
<li class="active"><a href="#">Home</a></li>
<li><a href="#">HTML</a></li>
<li><a href="#">CSS</a></li>
<li><a href="#">PHP</a></li>
<li><a href="#">Java</a></li>
</ul>
```

### .nav-fill

Instead of justifying, you can also force your menu items to fill in all the available space using this command. However, all the items will not have the same width.

```
<ul class="nav nav-pills nav-fill">
<li class="nav-item">
  <a class="nav-link active" href="#!">Active</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="#!">This link is looooong</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="#!">Short Link</a>
</li>
</ul>
```

# Navbar

Navbar is a responsive navigation header with lots of flexibility and support for branding, forms, links, and more.

## NavBar – basic navigation headers at top of page

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">My first navbar</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link 1</a>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled" href="#">Link Disabled</a>
      </li>
    </ul>
    <form class="form-inline my-2 my-lg-0">
      <input class="form-control mr-sm-2" type="text" placeholder="Search" aria-label="Search">
      <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Find Stuff</button>
    </form>
  </div>
</nav>
```

### .navbar-brand

Navbars come pre-furnished with support for some sub-components. This element will make your text stand out more. It was pre-designed to accommodate a product or company name.

```
<nav class="navbar navbar-light bg-light">
  <a class="navbar-brand" href="#">Your Company Name</a>
</nav>
```

### .navbar-text

Use this class to add centered text strings vertically and horizontal spacing

```
<nav class="navbar navbar-light bg-light">
  <span class="navbar-text">
    I'm a navbar text with an inline element
  </span>
</nav>
```

### .navbar-expand (-sm -md -lg -xl)

Enable responsive collapsing.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <button class="navbar-toggler navbar-toggler-right" type="button" data-toggle="collapse" data-target="#navbarTogglerDemo02" aria-controls="navbarTogglerDemo02" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <a class="navbar-brand" href="#">Navbar</a>
  <div class="collapse navbar-collapse" id="navbarTogglerDemo02">
    <ul class="navbar-nav mr-auto mt-2 mt-md-0">
```

```
<li class="nav-item active">
  <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
</li>
<li class="nav-item">
  <a class="nav-link" href="#">Link</a>
</li>
</ul>
</div>
</nav>
```

### .navbar-toggler

Program different navigation toggling behaviors (e.g. keep items active or collapse them).

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <button class="navbar-toggler navbar-toggler-right" type="button" data-
  toggle="collapse" data-target="#navbarTogglerDemo02" aria-controls="navbarTogglerDemo02"
  aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <a class="navbar-brand" href="#">Navbar</a>

  <div class="collapse navbar-collapse" id="navbarTogglerDemo02">
    <ul class="navbar-nav mr-auto mt-2 mt-md-0">
      <li class="nav-item active">
        <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
    </ul>
  </div>
</nav>
```

### .form-inline

Place a form (e.g. search bar) in nav heading.

```
<nav class="navbar navbar-light bg-light">
  <form class="form-inline">
    <input class="form-control mr-sm-2" type="text" placeholder="Search">
    <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>
  </form>
</nav>
```

### .collapse.navbar-collapse

Group and collapse navbar contents by a parent breakpoint.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
    </ul>
  </div>
</nav>
```

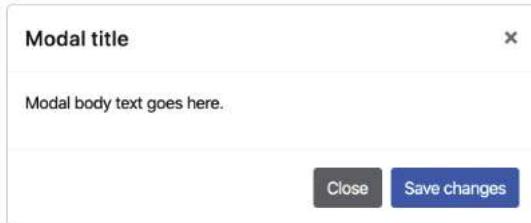
.navbar fixed-top

Set a fixed nav bar as primary and in dark color

```
<nav class="navbar fixed-top navbar-dark bg-primary">
  <a class="navbar-brand" href="#">Fixed top</a>
</nav>
```

## Bootstrap Modal (Plugin)

This is a JavaScript plugin used to add dialogs to a site, such as user notifications, custom content, and lightbox popups.



Modals consist of HTML, CSS, and JavaScript. And you can display one modal window at a time only, as Bootstrap creators deem nested models as a poor UX practice.

### Modal Example

```
<div class="modal" tabindex="-1" role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">Give it a title</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
        <p>Add some modal body text here.</p>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-primary">Agree</button>
        <button type="button" class="btn btn-secondary" data-
dismiss="modal">Disagree</button>
      </div>
    </div>
  </div>
</div>
```

### .modal-dialog-centered

You can choose to center all content vertically.

```
<div class="modal" tabindex="-1" role="dialog">
  <div class="modal-dialog modal-dialog-centered" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalCenterTitle">Modal title</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
```

List some texts.

```
</div>
<div class="modal-footer">
  <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
  <button type="button" class="btn btn-primary">Save changes</button>
</div>
</div>
</div>
```

### .modal-dialog-scrollable

When you need to pack in more content into a modal (e.g. a lengthy privacy policy), you can add this class to make it scroll it independently from the page.

```
<div class="modal" id="exampleModalScrollable" tabindex="-1" role="dialog" aria-labelledby="exampleModalScrollableLabel" aria-hidden="true">
  <div class="modal-dialog modal-dialog-scrollable" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalCenteredLabel">Modal title</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
```

This is a super long terms & conditions agreement that you'll agree to without actually reading it.

```
    </div>
    <div class="modal-footer">
      <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
      <button type="button" class="btn btn-primary">Save changes</button>
    </div>
  </div>
</div>
```

### .modal fade

Enabled fading for the content.

```
<div class="modal fade" id="exampleModal3" tabindex="-1" role="dialog" aria-labelledby="exampleModal3Label" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModal3Label">Modal title</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
```

Gotta right at least something here.

```
    </div>
    <div class="modal-footer">
      <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
      <button type="button" class="btn btn-primary">Save changes</button>
    </div>
  </div>
</div>
```

### .modal-lg, -sm, -xl

You can also adjust the sizes of your modals.

```
<!-- Large modal -->
<button class="btn btn-primary" data-toggle="modal" data-target=".bd-example-modal-lg">
  Large modal
</button>
<div class="modal fade bd-example-modal-lg" tabindex="-1" role="dialog" aria-labelledby="myLargeModalLabel" aria-hidden="true">
  <div class="modal-dialog modal-lg">
    <div class="modal-content">
      I'm a large modal!
    </div>
  </div>
</div>
```

```

</div>
</div>
<!-- Small modal -->
<button type="button" class="btn btn-primary" data-toggle="modal" data-target=".bd-example-modal-sm">
  Small modal
</button>

<div class="modal fade bd-example-modal-sm" tabindex="-1" role="dialog"
  aria-labelledby="mySmallModalLabel" aria-hidden="true">
  <div class="modal-dialog modal-sm">
    <div class="modal-content">
      I'm a small and cute modal.
    </div>
  </div>
</div>
<!-- Huge modal -->
<button type="button" class="btn btn-primary" data-toggle="modal" data-target=".bd-example-modal-xl">
  Huge modal
</button>

<div class="modal fade bd-example-modal-xl" tabindex="-1" role="dialog"
  aria-labelledby="myHugeModalLabel" aria-hidden="true">
  <div class="modal-dialog modal-xl">
    <div class="modal-content">
      I'm a jumbo-sized modal!
    </div>
  </div>
</div>

```

NOTE: Bootstrap offers a demo of all of these components at the Modal page. They are worth a look.

## Paginators

If you know HTML, paginators are nothing new to you. With Bootstrap you have several options for styling them.

```
.pagination - basic pagination example
<nav aria-label="Page navigation example">
  <ul class="pagination">
    <li class="page-item">
      <a class="page-link" href="#" aria-label="Previous">
        <span aria-hidden="true">&lt;</span>
        <span class="sr-only">Previous</span>
      </a>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item">
      <a class="page-link" href="#" aria-label="Next">
        <span aria-hidden="true">&gt;></span>
        <span class="sr-only">Next</span>
      </a>
    </li>
  </ul>
</nav>
```

### .page-item disabled

You can choose to disable one of the pagination elements or several ones.

Previous	1	2	3	Next
----------	---	---	---	------

```
<nav aria-label="...>
  <ul class="pagination">
    <li class="page-item disabled">
      <a class="page-link" href="#" tabindex="-1" aria-disabled="true">Previous</a>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item active" aria-current="page">
      <a class="page-link" href="#">2 <span class="sr-only">(current)</span></a>
    </li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item">
      <a class="page-link" href="#">Next</a>
    </li>
  </ul>
</nav>
```

### .page-item active

active indicates the current stage by highlighting it with blue.

```
<nav aria-label="...>
  <ul class="pagination">
    <li class="page-item disabled">
      <a class="page-link" href="#" tabindex="-1">Previous</a>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item active">
      <a class="page-link" href="#">2 <span class="sr-only">(current)</span></a>
    </li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item">
      <a class="page-link" href="#">Next</a>
    </li>
  </ul>
</nav>
```

### .pagination-lg

Make your pagination bigger and bolder.

```
<nav aria-label="...>
  <ul class="pagination pagination-lg">
    <li class="page-item disabled">
      <a class="page-link" href="#" tabindex="-1">Previous</a>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item">
      <a class="page-link" href="#">Next</a>
    </li>
  </ul>
</nav>
```

### .pagination-sm

Or make it petite and less visible in size.

```
<nav aria-label="...">
  <ul class="pagination pagination-sm">
    <li class="page-item disabled">
      <a class="page-link" href="#!" tabindex="-1">Previous</a>
    </li>
    <li class="page-item"><a class="page-link" href="#!">1</a></li>
    <li class="page-item"><a class="page-link" href="#!">2</a></li>
    <li class="page-item"><a class="page-link" href="#!">3</a></li>
    <li class="page-item">
      <a class="page-link" href="#!">Next</a>
    </li>
  </ul>
</nav>
```

## Popovers

Popover plugin enables you to create a pop-up box with content and other elements, activated whenever a user clicks on the element. Popovers are similar to tooltips, but fit more content.

Things to Know Before You Begin:

- You have to add the attribute, `data-toggle="popover"` to an element to create a popover.
  - You must use the `title` attribute to specify header text
  - Use the `data-content` attribute to specify what content is to be displayed in the body of the popover.
- ```
<a href="#" data-toggle="popover" title="Popover Header" data-content="Some content inside the popover">Toggle popover</a>
```

You have to use jQuery to initialize popovers - `popover()`

### Popover

Here's the code that will enable popovers on your page:

```
<script>
$(document).ready(function(){
  $('[data-toggle="popover"]').popover();
});
</script>
```

Positioning Your Popovers - top, bottom, left, or right

Specify one of the four positions for your popover.

Popover on top

```
<button type="button" class="btn btn-secondary" data-container="body" data-
toggle="popover" data-placement="top" data-content="Vivamus sagittis lacus vel augue
laoreet rutrum faucibus.">
```

Popover on top

```
</button>
```

Popover on right

```
<button type="button" class="btn btn-secondary" data-container="body" data-
toggle="popover" data-placement="right" data-content="Vivamus sagittis lacus vel
augue laoreet rutrum faucibus.">
```

Popover on right

```
</button>
```

Popover on bottom

```
<button type="button" class="btn btn-secondary" data-container="body" data-
toggle="popover" data-placement="bottom" data-content="Vivamus
sagittis lacus vel augue laoreet rutrum faucibus.">
```

Popover on bottom

```
</button>
```

Popover on left

```
<button type="button" class="btn btn-secondary" data-container="body" data-
toggle="popover" data-placement="left" data-content="Vivamus sagittis lacus vel
augue laoreet rutrum faucibus.">
```

Popover on left

```
</button>
```

## Creating a Container Popover

Code a custom container whenever you feel that some styles on a parent element can mess up its look. Adding a container attribute will "wrap" the popover's HTML within the parent element.

```
$(function () {
  $('.example-popover').popover({
    container: 'body'
  })
})
```

### .popover-dismiss

Provide users with the option to close a popover when they click on the element the second time. You can arrange for the popover to dismiss when user clicks outside the element too: Use the data-trigger="focus" attribute.

By default, the popover is closed when you click on the element again. However, you can use the data-trigger="focus" attribute which will close the popover when clicking outside the element:

```
<a href="#" title="Dismissible popover" data-toggle="popover" data-trigger="focus"
data-content="Click anywhere in the document to close this popover">Click me</a>
```

# Progress

Develop a custom progress bar and add additional styling elements such as animation and text labels if you like.

### .progress

The class for setting up a basic progress bar. It acts as a wrapper, indicating the max value of your progress bar.

```
<div class="progress">
  <div class="progress-bar" role="progressbar" style="width: 41%" aria-valuenow="41"
  aria-valuemin="0" aria-valuemax="100"></div>
</div>
```

### .progress-bar

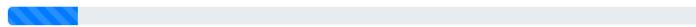
Use this command to specify the current progress.

```
<div class="progress">
  <div class="progress-bar" role="progressbar" style="width: 55%" aria-valuenow="55"
  aria-valuemin="0" aria-valuemax="100"></div>
</div>
```

### .progress-bar-striped

Add some stripes to the progress bar section.

```
<div class="progress">
  <div class="progress-bar progress-bar-striped" role="progressbar" style="width: 10%" aria-valuenow="10" aria-valuemin="0" aria-valuemax="100"></div>
</div>
```



### .progress-bar-animated

No get those stripes moving, barber's pole style.

```
<div class="progress">
  <div class="progress-bar progress-bar-striped progress-bar-animated" role="progressbar" aria-valuenow="75" aria-valuemin="0" aria-valuemax="100" style="width: 75%"></div>
</div>
```

## Add Multiple Progress Bars

You can simultaneously add several bars to indicate progress for different elements. Adjust the allocation for each one up to your liking. .



```
<div class="progress">
  <div class="progress-bar" role="progressbar" style="width: 15%" aria-valuenow="15" aria-valuemin="0" aria-valuemax="100"></div>
  <div class="progress-bar bg-success" role="progressbar" style="width: 30%" aria-valuenow="30" aria-valuemin="0" aria-valuemax="100"></div>
  <div class="progress-bar bg-info" role="progressbar" style="width: 20%" aria-valuenow="20" aria-valuemin="0" aria-valuemax="100"></div>
</div>
```

## Spinner

Use this plugin to add that animated spinner. Spinners are built with HTML and CSS only, without any JS. But...you may need JavaScript for some styling options e.g. to toggle their visibility. Also, you have several nice options to choose from.

.spinner-border

Build a black and white loading indicator.

```
<div class="spinner-border" role="status">
  <span class="sr-only">Loading...</span>
</div>
```

.spinner-border-primary (-secondary, -light, -dark, -warning, -success, -info, -danger)

Or add a colorful border, using the standard Bootstrap styling attributes.

```
<div class="spinner-border text-primary" role="status"> <span class="sr-
only">Loading...</span> </div>
<div class="spinner-border text-secondary" role="status"> <span class="sr-
only">Loading...</span> </div>
<div class="spinner-border text-success" role="status"> <span class="sr-
only">Loading...</span> </div>
<div class="spinner-border text-danger" role="status"><span class="sr-
only">Loading...</span> </div>
<div class="spinner-border text-warning" role="status"> <span class="sr-
only">Loading...</span> </div>
<div class="spinner-border text-info" role="status"><span class="sr-
only">Loading...</span> </div>
<div class="spinner-border text-light" role="status"><span class="sr-
only">Loading...</span></div>
<div class="spinner-border text-dark" role="status"><span class="sr-
only">Loading...</span> </div>
```

Growing Spinner

Create a pulsating spinner that increases and reduces in size. A funky alternative to the spinning spinner :). Again, it can be designed in several colors.

```
<div class="spinner-grow text-primary" role="status"> <span class="sr-
only">Loading...</span> </div>
```

## Table

Bootstrap offers an easy-peasy way to create tables. Add base class .table to an `<table>` and add-on extra custom styles.

NB: All table styles are inherited in Bootstrap 4. Every nested table will be styled just as the parent.

```
.table - example
<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Mark</td>
      <td>Otto</td>
      <td>Qmdo</td>
    </tr>
    <tr>
```

```

<th scope="row">2</th>
<td>Jacob</td>
<td>Thornton</td>
<td>@fat</td>
</tr>
<tr>
  <th scope="row">3</th>
  <td>Larry</td>
  <td>the Bird</td>
  <td>@twitter</td>
</tr>
</tbody>
</table>

```

### .table-dark

Create a table with a dark background and light texts.

```

<table class="table table-dark">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">Nickname</th>
      <th scope="col">Followers</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Minimark</td>
      <td>5,000</td>
      <td>@minik</td>
    </tr>
    <tr>
      <th scope="row">2</th>
      <td>Gigi_B</td>
      <td>24K</td>
      <td>@gig</td>
    </tr>
    <tr>
      <th scope="row">3</th>
      <td>Birdie</td>
      <td>50K</td>
      <td>@birdie</td>
    </tr>
  </tbody>
</table>

```

### .thead-light and .thead-dark

You can also choose to highlight only one of the table entries using the dark/light attribute.

```

<table class="table">
  <thead class="thead-dark">
    <tr>
      <th scope="col">#</th>
      <th scope="col">Episode</th>
      <th scope="col">Name</th>
      <th scope="col">Rating</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>#4</td>
      <td>New Hope</td>
      <td>10/10</td>
    </tr>
  </tbody>
</table>

```

```

<tr>
  <th scope="row">2</th>
  <td>#6</td>
  <td>Return of the Jedi</td>
  <td>9/10</td>
</tr>
<tr>
  <th scope="row">3</th>
  <td>#9</td>
  <td>Rise of the Skywalker</td>
  <td>6/10</td>
</tr>
</tbody>
</table>






```

**.table-striped**  
Add this class to within the `<tbody>` to make your table zebra-stripey.

```

<table class="table table-striped">
  <thead>
    <tr>
      <th>#</th>
      <th>First Name</th>
      <th>Last Name</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Darth</td>
      <td>Vader</td>
    </tr>
    <tr>
      <th scope="row">2</th>
    </tr>
    <tr>
      <th scope="row">3</th>
      <td>Leia</td>
      <td>Organa</td>
    </tr>
    <tr>
      <th scope="row">4</th>
      <td>Han</td>
      <td>Solo</td>
    </tr>
    <tr>
      <th scope="row">5</th>
      <td>Luke</td>
      <td>Skywalker</td>
    </tr>
    <tr>
      <th scope="row">6</th>
      <td>Obi-Wan</td>
      <td>Kenobi</td>
    </tr>
    <tr>
      <th scope="row">7</th>
      <td>Yoda</td>
      <td>Jedi</td>
    </tr>
    <tr>
      <th scope="row">8</th>
      <td>Palpatine</td>
      <td>Dark Lord</td>
    </tr>
    <tr>
      <th scope="row">9</th>
      <td>Leia</td>
      <td>Organa</td>
    </tr>
    <tr>
      <th scope="row">10</th>
      <td>Leia</td>
      <td>Skywalker</td>
    </tr>
  </tbody>

```

```

<td>Ray</td>
<td>Just Ray</td>
</tr>
</tbody>
</table>

```

### .table-bordered

Style borders for all the cells and table sides.

```

<table class="table table-bordered">
  <thead>
    <tr>
      <th>#</th>
      <th>First Name</th>
      <th>Last Name</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Lil</td>
      <td>Kim</td>
    </tr>
    <tr>
      <th scope="row">2</th>
      <td>Kanye</td>
      <td>West</td>
    </tr>
  </tbody>
</table>

```

### .table-borderless

Ditch the borders to give your table a minimalistic flair.

```

<table class="table table-borderless">
  <thead>
    <tr>
      <th>#</th>
      <th>First Name</th>
      <th>Last Name</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Indiana</td>
      <td>Jones</td>
    </tr>
    <tr>
      <th scope="row">2</th>
      <td>Han</td>
      <td>Solo</td>
    </tr>
  </tbody>
</table>

```

### .table-hover

Add a hover state on table rows.

```

<table class="table table-hover">
  <thead>
    <tr>
      <th>#</th>
      <th>First Name</th>
      <th>Last Name</th>
    </tr>
  </thead>

```

```

</thead>
<tbody>
  <tr>
    <th scope="row">1</th>
    <td>Mark</td>
    <td>Twain</td>
  </tr>
  <tr>
    <th scope="row">2</th>
    <td>Henry</td>
    <td>James</td>
  </tr>
</tbody>
</table>

```

### .table-sm

Minify your table. This class will cut the cell padding in half.

```

<table class="table table-sm">
  <thead>
    <tr>
      <th>#</th>
      <th>First Name</th>
      <th>Last Name</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Amy</td>
      <td>Winehouse</td>
    </tr>
    <tr>
      <th scope="row">2</th>
      <td>Kurt</td>
      <td>Cobain</td>
    </tr>
  </tbody>
</table>

```

### .table-active (-primary, -default, -secondary, -success, -danger, -warning, -info, -light, -dark).

Pain individual table rows in different colors.

```

<tr class="table-active">...</tr>
<tr class="table-primary">...</tr>
<tr class="table-secondary">...</tr>
<tr class="table-success">...</tr>
<tr class="table-danger">...</tr>
<tr class="table-warning">...</tr>
<tr class="table-info">...</tr>
<tr class="table-light">...</tr>
<tr class="table-dark">...</tr>

```

## Toasts

A plugin that allows you to add “push notifications” with Flexbox - very easy to position and align. use these as an alternative to alerts.

```
.toast - example
<div class="toast" role="alert" aria-live="assertive" aria-atomic="true">
  <div class="toast-header">
    
    <strong class="mr-auto">Bootstrap</strong>
    <small>11 mins ago</small>
    <button type="button" class="ml-2 mb-1 close" data-dismiss="toast" aria-label="Close">
      <span aria-hidden="true">&times;</span>
    </button>
  </div>
  <div class="toast-body">
    Kudos, I'm a toast message!
  </div>
</div>
```

## Tooltips

Tooltips are small text pop-ups that provide users with some additional context on the button or another website element. In Bootstrap 4, tooltips use Popper.js library for positioning. That's why to use them you must include popper.min.js before bootstrap.js or use bootstrap.bundle.min.js / bootstrap.bundle.js.

### Enable Tooltips

```
$(function () {$('.[data-toggle="tooltip"]').tooltip()})
```

### Align Tool Tips

#### Tooltip on top

```
<button type="button" class="btn btn-secondary" data-toggle="tooltip" data-placement="top" title="Tooltip on top"> </button>
```

#### Tooltip on right

```
<button type="button" class="btn btn-secondary" data-toggle="tooltip" data-placement="right" title="Tooltip on right"></button>
```

#### Tooltip on bottom

```
<button type="button" class="btn btn-secondary" data-toggle="tooltip" data-placement="bottom" title="Tooltip on bottom"></button>
```

#### Tooltip on left

```
<button type="button" class="btn btn-secondary" data-toggle="tooltip" data-placement="left" title="Tooltip on left">
```

For more advanced customizations and JavaScript methods, check the official Tooltip documentation.

# Bootstrap Styling Essentials

## Breakpoints

For most components, layouts and grid systems Bootstrap uses the following breakpoint values:

Extra small < 544px

Small ≥ 544px

Medium ≥ 768px

Large ≥ 992px

Extra large ≥ 1200px

### Grids and Columns

Easy to layout and align content through Flexbox; totally responsive. You get a 12-column system, 5 tiers by default, predefined classes and lots of variables.

### Grid Sizing Options

- .col-xs-: container width - auto/none; column width - auto.
- .col-sm-: container width - 750px; column width - ~62px
- .col-md-: container width - 970px ; column width - ~81px
- .col-lg-: container width - 1170px; column width - ~97px

### Basic .container parent grid

Create 3 equal columns with predefined grid classes

```
<div class="container">
  <div class="row">
    <div class="col-sm">
      One of three columns
    </div>
    <div class="col-sm">
      One of three columns
    </div>
    <div class="col-sm">
      One of three columns
    </div>
  </div>
</div>
```

### .container-fluid

Scale your container to span over the entire viewpoint.

```
<div class="container-fluid">
  <!-- A fluid container that uses the full width -->
</div>
```

### Column Sizes

Select among xs, sm, md, lg. Again, when you add columns, they have to fit the screen sizes of all devices or users will be frustrated

Here's the code snippet - just change the size element

```
<div class="container">
  <div class="row">
    <div class="col-xl-2">
      <!-- Content -->
```

```

</div>
<div class="col-xl-2">
    <!-- Content -->
</div>
<div class="col-xl-8">
    <!-- Content -->
</div>
</div>
</div>

```

### .no-gutters

The standard space between columns is 15px on each side. But you can remove it.

```

<div class="container">
    <div class="row no-gutters">
        <div class="col-12 col-sm-6 col-md-8".col-12 .col-sm-6 .col-md-8</div>
        <div class="col-6 col-md-4".col-6 .col-md-4</div>
    </div>
</div>

```

## Typography: Everything you need for headings, body text, lists, and much more.

### Text Alignment – left, right, centered, justified

```

<div> <p class="text-left">dotnetfunda is an online tutorial</p>
    <p class="text-center">dotnetfunda is an online tutorial</p>
    <p class="text-right">dotnetfunda is an online tutorial</p> </div>

```

### List inline – display a list in a line

```

<ul class="list-inline">
    <li>Tea</li>
    <li>Sugar</li>
    <li>Milk</li>
</ul>

```

### List-unstyled – remove the default list style.

```

<ul class="list-unstyled">
    <li>Coffee</li>
    <li>
        Tea
        <ul>
            <li>Black tea</li>
            <li>Green tea</li>
        </ul>
    </li>
    <li>Milk</li>
</ul>

```

### .text-lowercase

```
<p class="text-lowercase">lowercased text.</p>
```

### .text-uppercase

```
<p class="text-uppercase">uppercase text.</p>
```

```
.text-capitaliz
<p class="text-capitaliz">capitalized text.</p>

.text-truncate
<p class="text-truncate">truncated text.</p>

.text-nowrap
<p class="text-nowrap">No Wrap Text.</p>

.text-monospace
<p class="text-monospace">This is in monospace</p>

.text-hide
<h1 class="text-hide">Custom heading</h1>

.text-decoration-none
<p class="text-decoration-none">this text is not de
```

1

When you want to add emphasis to a specific paragraph of content, you can make the font a bit larger and even thinner, so that the paragraph stands out visually.

**<h2 class="lead">This is the info you should really pay attention to</h2>**

Blockquote

If you are inserting a quote that is a bit lengthy, this will class allow you to set it off from the rest of the text.

```
<blockquote class="blockquote">  
  <p class="mb-0">Some cool quote that you've found</p>  
</blockquote>
```

Blockquote Footer

Add the source of a quote by adding the footer element

```
<blockquote class="blockquote">
  <p class="mb-0">Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
  <footer class="blockquote-footer">Someone famous in
    <cite title="Source Title">Source Title</cite>
  </footer>
</blockquote>
```

#### `.text-break`

This prevents super-long texts from breaking the look of your other design elements.

## Floats

### .float-left - floats item left - all sizes

```
<div class="float-left">Float left on all viewport sizes</div>
<div class="float-sm-left">Float left on viewports sized SM (small) or wider</div>
<div class="float-md-left">Float left on viewports sized MD (medium) or wider</div>
<div class="float-lg-left">Float left on viewports sized LG (large) or wider</div>
<div class="float-xl-left">Float left on viewports sized XL (extra-large) or wider</div>
```

### .float-right - Floats item right - all sizes

```
<div class="float-right">Float right on all viewport sizes</div>
<div class="float-sm-right">Float right on viewports sized SM (small) or wider</div>
<div class="float-md-right">Float right on viewports sized MD (medium) or wider</div>
<div class="float-lg-right">Float right on viewports sized LG (large) or wider</div>
<div class="float-xl-right">Float right on viewports sized XL (extra-large) or wider</div>
```

### .float-none - Removes float - all sizes

```
<div class="float-none">Don't float on all viewport sizes</div>
<div class="float-sm-none">Don't float on viewports sized SM (small) or wider</div>
<div class="float-md-none">Don't float on viewports sized MD (medium) or wider</div>
<div class="float-lg-none">Don't float on viewports sized LG (large) or wider</div>
<div class="float-xl-none">Don't float on viewports sized XL (extra-large) or wider</div>
```

### .clearfix - Removes float from parent element

```
<div class="clearfix">...</div>
```

## Flex

You can easily and quickly manage your layouts, alignments, sizings, navs, displays, colors, alignment, and much more with this single Bootstrap utility tool.

### Flex - sm-lg-md-xl

Size all of your elements using predetermined sizes and be fully responsive.

```
<div class="d-flex flex-sm-column">...</div>
<div class="d-flex flex-md-column">...</div>
<div class="d-flex flex-lg-column">...</div>
<div class="d-flex flex-xl-column">...</div>
```

### .flex-row and .flex-row-reverse

Set horizontal direction or reverse it to start from the opposite side.

```
<div class="d-flex flex-row">
  <div class="p-2">Flex item 1</div>
  <div class="p-2">Flex item 2</div>
  <div class="p-2">Flex item 3</div>
</div>
```

```
<div class="d-flex flex-row-reverse">
  <div class="p-2">Flex item 1</div>
  <div class="p-2">Flex item 2</div>
  <div class="p-2">Flex item 3</div>
</div>
```

### .flex-column and .flex-column-reverse

Set direction of columns or reverse to set from opposite direction.

```
<div class="d-flex flex-column">
  <div class="p-2">Flex item 1</div>
  <div class="p-2">Flex item 2</div>
  <div class="p-2">Flex item 3</div>
</div>

<div class="d-flex flex-column-reverse">
  <div class="p-2">Flex item 1</div>
  <div class="p-2">Flex item 2</div>
  <div class="p-2">Flex item 3</div>
</div>
```

### .Flex-fill

This utility forces all elements to occupy an equal width while taking up all the available horizontal space.

```
<div class="d-flex text-white">
  <div class="p-2 flex-fill bg-primary">Flex item</div>
  <div class="p-2 flex-fill bg-success">Flex item</div>
  <div class="p-2 flex-fill bg-primary">Flex item</div>
</div>
```

### .Flex-grow-\*

One of the flex items will fill all the available space, while ensuring that others also have minimally sufficient space.

```
<div class="d-flex text-white">
  <div class="p-2 flex-grow-1 bg-primary">Flex item</div>
  <div class="p-2 bg-success">Flex item</div>
  <div class="p-2 bg-primary">Third flex item</div>
</div>
```

### .Flex-shrink-\*

Activate the item's ability to shrink to the most minimal size to fill in the available space.

```
<div class="d-flex text-white">
  <div class="p-2 w-100 bg-primary">Flex item</div>
  <div class="p-2 flex-shrink-1 bg-success">Flex item</div>
</div>
```

### .flexnowrap, .flex-wrap-reverse and .flex-wrap,

Use these classes when you want items either to wrap to the next line, or to fit on a single line (nowrap), or do the reverse wrapping act. (wrap-reverse).

```
<div class="d-flex flexnowrap"> ... </div>
<div class="d-flex flex-wrap"> ... </div>
<div class="d-flex flex-wrap-reverse"> ... </div>
```

### .flex justify-content

Add this class when you want to justify text on right or left, and when you want that justification to start and end.

```
<div class="d-flex justify-content-start">
  <div class="p-2">Flex item 1</div>
  <div class="p-2">Flex item 2</div>
  <div class="p-2">Flex item 3</div>
</div>

<div class="d-flex justify-content-end">
  <div class="p-2">Flex item 1</div>
  <div class="p-2">Flex item 2</div>
  <div class="p-2">Flex item 3</div>
</div>
```

### .flex align-items

Specify when the alignment begins and when it ends for a list of items.

```
<div class="d-flex align-items-start">
  <div class="p-2">Flex item 1</div>
  <div class="p-2">Flex item 2</div>
  <div class="p-2">Flex item 3</div>
</div>

<div class="d-flex align-items-end">
  <div class="p-2">Flex item 1</div>
  <div class="p-2">Flex item 2</div>
  <div class="p-2">Flex item 3</div>
</div>
```

### .flex align-self-start

You can also override a containers alignment for a selected item within that container

```
<div class="align-self-start">Aligned flex item</div>
```

## Alignment

How and where do you want to align your content, buttons, etc. - right, left, center?

```
.align-baseline
<div class="align-self-baseline">Aligned flex item</div>
```

```
.align-top -middle - bottom: place buttons where you want.
<span class="align-baseline">baseline</span>
<span class="align-top">top</span>
<span class="align-middle">middle</span>
<span class="align-bottom">bottom</span>
```

```
.align-text-top -bottom: place text where you want
<span class="align-text-top">text-top</span>
<span class="align-text-bottom">text-bottom</span>
```

## Borders

Borders add elegance to a site. Use them for emphasis or aesthetics

```
.border
<span class="border border-dark">Hello World</span>
```

```
.border-light, -dark -primary, -secondary, -warning, -success,
-danger, -info. -white)
<span class="border border-light"></span>
<span class="border border-dark"></span>
<span class="border border-primary"></span>
<span class="border border-secondary"></span>
<span class="border border-warning"></span>
<span class="border border-success"></span>
<span class="border border-danger"></span>
<span class="border border-info"></span>
<span class="border border-white"></span>
```

### Border- (top, right, bottom, left,)

```
<span class="p-1 border border-primary border-0">Hello World</span>
<span class="p-1 border border-primary border-top-0">Hello World</span>
<span class="p-1 border border-primary border-right-0">Hello World</span>
<span class="p-1 border border-primary border-bottom-0">Hello World</span>
<span class="p-1 border border-primary border-left-0">Hello World</span>
```

### Border Corner Styling Options



```







```

## Position- configure the position of elements

### .sticky-top

Position an element at the top of viewport, going from edge to edge.

```
<div class="sticky-top bg-primary">Sticky top</div>
```

### .fixed-top, .fixed-bottom

Place an element at the top or bottom of the viewport, from edge to edge.

```
<div class="fixed-top">Something atop</div>
<div class="fixed-bottom">Something at the bottom</div>
```

## Shadows - add decorative contrasts for extra appeal.

### .shadow

```
<div class="shadow p-3 mb-5 bg-light rounded">Regular shadow</div>
```

### .shadow-none

```
<div class="shadow-none p-3 mb-5 bg-light rounded">No shadow</div>
```

### .shadow-sm

```
<div class="shadow-sm p-3 mb-5 bg-light rounded">Small Shadow shadow</div>
```

### .shadow-lg

```
<div class="shadow-lg p-3 mb-5 bg-white rounded">Larger shadow</div>
```

## Visibility - tune the visibility of any element

### Visible

```
<div class="visible">...</div>
```

### Invisible

```
<div class="invisible">...</div>
```



# JavaScript

## CheatSheet



In this Cheatsheet, we will cover the basics of JavaScript. We will provide examples to help you understand how JavaScript work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.

# JS **JavaScript**

JavaScript is a programming language that is widely used in web development. It is a client-side scripting language, which means that it is executed on the client side (in a web browser) rather than on the server side. JavaScript is used to create interactive web pages and is an essential component of modern web development. It is a high-level, dynamically typed language that is interpreted, which means that it is not compiled into machine code before it is run. JavaScript is used to add functionality to websites, such as handling user events (such as clicks and form submissions), animating page elements, and making asynchronous requests to servers. It is a versatile language that can be used for a wide range of applications, from simple scripts that add simple interactivity to websites to complex single-page applications.

## On page script:

```
<script type="text/javascript"> .... </script>
```

## Include external JS file:

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

## Delay - 1 second timeout:

```
setTimeout(function () {  
}, 1000);
```

## Functions:

```
function addNumbers(a, b) {  
    return a + b;  
}  
  
x = addNumbers(1, 2);
```

## Edit DOM element:

```
document.getElementById("elementID").innerHTML = "Hello World!";
```

## Output:

```
console.log(a);      // write to the browser console.  
document.write(a);  // write to the HTML.  
alert(a);          // output in an alert box.  
confirm(?);        // yes/no dialog, returns true/false depending on user click.  
prompt("Age ?","0"); // input dialog box. (Initial Value = 0).
```

## Comments:

```
/* Multi line comment */  
// One line
```

## Variables:

```
var v;                      // variable  
var b = "CodeHelp";          // string  
var c = "Code" + " " + "Help"; // = "Code Help"  
var d = 1 + 4 + "5";         // = "55"  
var e = [1,2,6,8];           // array  
var f = true;                // boolean  
var g = /()/;                // RegEx  
var h = function(){};         // function object  
const PI = 3.14;              // constant  
var a = 7, b = 6, c = a + b;   // one line  
let a = 'aaa';                // block scope local variable
```

## Strict mode:

```
"use strict";               // Use strict mode to write secure code  
x = 1;                      // Throws an error because variable is not declared
```

## Values:

```
false, true                                // Boolean  
4, 3.14, 0b10011, 0xF6, NaN                // Number  
"CodeHelp", 'Love'                          // String  
undefined, Infinity, null                  // Special
```

## Basic Operators:

```
a = b + c - d;                      // Addition, Subtraction.  
a = b * (c / d);                    // Multiplication, Division.  
x = 10 % 4;                        // Modulo, 10 / 4 Remainder = 2.  
a++; b--;                           // Postfix Increment and Decrement.
```

## Bitwise Operators:

&	AND	5 & 1	(0101 & 0001)	1	(1)
	OR	5   1	(0101   0001)	5	(101)
~	NOT	~ 5	(~0101)	10	(1010)
^	XOR	5 ^ 1	(0101 ^ 0001)	4	(100)
<<	Left Shift	5 << 1	(0101 << 1)	10	(1010)
>>	Right Shift	5 >> 1	(0101 >> 1)	2	(10)
>>>	Zero Fill Right Shift	5 >>> 1	(0101 >>> 1)	2	(10)

## Arithmetic Operators:

<code>x * (y + z)</code>	// grouping
<code>x = 4</code>	// assignment
<code>x == y</code>	// equals
<code>x != y</code>	// unequal
<code>x === y</code>	// strict equal
<code>x !== y</code>	// strict unequal
<code>x &lt; y x &gt; y</code>	// less and greater than
<code>x &lt;= y x &gt;= y</code>	// less or equal, greater or equal
<code>x += y</code>	// $x = x + y$
<code>x &amp;&amp; y</code>	// logical AND
<code>x    y</code>	// logical OR
<code>!(x == y)</code>	// logical NOT
<code>x != y</code>	// not equal
<code>typeof x</code>	// type of x
<code>x &lt;&lt; 2 x &gt;&gt; 3</code>	// shifting

## Objects:

```
var student = { // object name
    firstName: "Koushik", // list of properties and values
    lastName: "Sadhu",
    age: 20,
    height: 175,
    fullName: function() { // object function
        return this.firstName + " " + this.lastName;
    }
};

student.age = 19; // setting value
student[age]++;
name = student.fullName(); // call object function
```

## Strings:

```
var a = "Codehelp";
var b = 'I don\'t \n know'; // \n new line.
var len = a.length; // string length.
a.indexOf("h"); // find substring, -1 if doesn't contain.
a.lastIndexOf("e"); // last occurrence.
a.slice(3, 6); // cut out "ehe", negative values count from behind.
a.replace("help","love"); // find and replace, takes regular expressions.
a.toUpperCase(); // convert to upper case.
a.toLowerCase(); // convert to lower case.
a.concat(" ", str2); // Codehelp + " " + str2.
a.charAt(4); // character at index 4: "h".
abc.split(" "); // splitting a string on space, and stores in an array.
```

## Numbers and Math:

```
var pi = 3.14;  
pi.toFixed(0);  
// returns 3  
pi.toFixed(2);  
// returns 3.14, working with money  
pi.toPrecision(2)  
// returns 3.1  
pi.valueOf();  
// returns number  
Number(true);  
// converts to number  
Number(new Date())  
// number of milliseconds since, 1970  
parseInt("3 months");  
// returns the first number 3  
parseFloat("3.5 days");  
// returns 3.5  
Number.MAX_VALUE  
// largest possible JS number  
Number.MIN_VALUE  
// smallest possible JS number  
Number.NEGATIVE_INFINITY  
// Minus Infinity  
Number.POSITIVE_INFINITY  
// Infinity  
var pi = Math.PI;  
// 3.141592653589793  
Math.round(4.5);  
// 5  
Math.pow(2,8);  
// 256 - 2 to the power of 8  
Math.sqrt(49);  
// 7 - square root  
Math.abs(-3.14);  
// 3.14 - absolute, positive value  
Math.ceil(3.14);  
// 4 - rounded up  
Math.floor(3.99);  
// 3 - rounded down  
Math.sin(0);  
// 0 - sine  
Math.cos(Math.PI);  
// OTHERS: tan, atan, asin, acos,  
Math.min(0, 3, -2, 2);  
// -2 the lowest value  
Math.max(0, 3, -2, 2);  
// 3 the highest value  
Math.log(1);  
// 0 natural logarithm  
Math.exp(1);  
// 2.7182pow(E,x)  
Math.random();  
// random number between 0 and 1  
Math.floor(Math.random() * 5) + 1;  
// random integer, from 1 to 5
```

## Array:

```
var flowers = ["Rose", "Sunflower", "Lotus", "Lily"];           // Array Declaration  
var flowers = new Array ("Rose", "Sunflower", "Lotus", "Lily"); //Alternate method.  
alert(flowers[1]);                                         // access value at index.  
dogs[0] = "Hibiscus";                                     // change the first item to "Hibiscus".  
for (var i = 0; i < flowers.length; i++) {                  // Accessing array element using loop.  
    console.log(flowers[i]);  
}
```

## Array Methods:

flowers.toString();	// convert the array to string.
flowers.join(" _ ");	// join between two array element.
flowers.pop();	// remove last element.
flowers.push("Tulip");	// add new element to the end.
flowers [flowers.length] = "Tulip";	// the same as push.
flowers.shift();	// remove first element.
flowers.unshift("Tulip");	// add new element to the beginning.
delete.flowers[0];	// change element to undefined.
flowers.splice(2, 0, "ABC", "DEF");	// add elements.
var f = flowers.concat(a,b);	// join two arrays (a followed by b).
flowers.slice(1,4);	// elements from [1] to [4-1].
flowers.sort();	// sort string alphabetically.
flowers.reverse();	// sort string in descending order.
x.sort(function(a, b){return a - b});	// numeric sort.
x.sort(function(a, b){return b - a});	// numeric descending sort.
x.sort(function(a, b){return 0.5 - Math.random()});	// random order sorting.

## Regular Expressions:

```
var a = str.search(/CheatSheet/i);
```

### Modifiers:

- **i** perform case-insensitive matching.
- **g** perform a global match.
- **m** perform multiline matching.

### Patterns:

- \ Escape character
- \d find a digit
- \s find a whitespace character
- \b find match at beginning or end of a word
- n+ contains at least one n
- n\* contains zero or more occurrences of n
- n? contains zero or one occurrences of n
- ^ Start of string
- \$ End of string
- \uxxxx find the Unicode character
- . Any single character
- (x | y) x or y
- ( ... ) Group section
- [xyz] In range (x, y or z)
- [0-9] any of the digits between the brackets
- [^xyz] Not in range
- \s White space

• <b>a?</b>	Zero or one of a
• <b>a*</b>	Zero or more of a
• <b>a*?</b>	Zero or more, ungreedy
• <b>a<sup>+</sup></b>	One or more of a
• <b>a<sup>+</sup>?</b>	One or more, ungreedy
• <b>a{2}</b>	Exactly 2 of a
• <b>a{2,}</b>	2 or more of a
• <b>a{,10}</b>	Up to 10 of a
• <b>a{1,6}</b>	1 to 6 of a
• <b>a{4,6}?</b>	4 to 6 of a
• <b>[:punct:]</b>	Any punctuation symbol
• <b>[:space:]</b>	Any space character
• <b>[:blank:]</b>	Space or tab

## If-Else Statements:

```

if ((age >= 10) && (age < 20)) {           // logical condition
    status = "Permitted.";
}                                               // executed if condition is true
else {
    status = "Not Permitted.";
}

```

// else block is optional

// executed if condition is false

## Switch Statement:

```
switch (day) {                                // Input is current day in numeric.  
    case 1:                                    // if (day == 1)  
        text = "Monday";  
        break;  
    case 2:                                    // if (day == 2)  
        text = "Tuesday";  
        break;  
    case 3:                                    // if (day == 3)  
        text = "Wednesday";  
        break;  
    case 4:                                    // if (day == 4)  
        text = "Thursday";  
        break;  
    case 5:                                    // if (day == 5)  
        text = "Friday";  
        break;  
    case 6:                                    // if (day == 6)  
        text = "Saturday";  
        break;  
    case 7:                                    // if (day == 7)  
        text = "Sunday";  
        break;  
    default:                                   // else...  
        text = "Please enter valid day number.";  
}
```

## For Loop:

```
for (var i = 0; i < 5; i++) {  
    document.write(i + ":" + i*i+ "<br/>");  
}
```

```
var sum = 0;  
for (var i = 0; i < a.length; i++) {  
    sum += a[i];  
}
```

```
html = "";  
for (var i of stud) {  
    html += "<li>" + i + "</li>";  
}
```

## While Loop:

```
var i = 1; // initialize  
while (i < 20) { // enters the cycle if statement is true  
    i += i; // increment to avoid infinite loop  
    document.write(i + ","); // output  
}
```

## Do While Loop:

```
var i = 1;           // initialize
do {
    i += 1;          // enters loop at least once
    document.write(i + ", ");
} while (i < 20)    // repeats loop, if statement is true
```

## Break Statement:

```
for (var i = 0; i < 20; i++) {
    if (i == 10) {
        break;           // terminates the loop
    }
    document.write(i + ", ");
}
```

## Continue Statement:

```
for (var i = 0; i < 20; i++) {
    if (i == 10) {
        continue;        // skips
    }
    document.write(i + ", ");
}
```

## Dates:

```
var d = new Date();
a = d.getDay();                                // getting the weekday
getDate();                                     // day as a number (1-31)
getDay();                                       // weekday as a number (0-6)
getFullYear();                                  // four digit year (yyyy)
getHours();                                     // hour (0-23)
getMilliseconds();                            // milliseconds (0-999)
getMinutes();                                   // minutes (0-59)
getMonth();                                     // month (0-11)
getSeconds();                                   // seconds (0-59)
getTime();                                      // milliseconds since 1970
d.setDate(d.getDate() + 7);                    // adds a week to a date
 setDate();                                     // day as a number (1-31)
setFullYear();                                 // year (optionally month and day)
setHours();                                    // hour (0-23)
setMilliseconds();                           // milliseconds (0-999)
setMinutes();                                   // minutes (0-59)
setMonth();                                     // month (0-11)
setSeconds();                                   // seconds (0-59)
 setTime();                                     // milliseconds since 1970
```

## Global Functions:

```
eval();                                // executes a string as if it was scriptcode  
String(44);                            // return string from number  
(44).toString();                      // return string from number  
Number("44");                          // return number from string  
decodeURI(enc);                        // decode URI. Result: "page.asp"  
encodeURI(uri);                        // encode URI. Result: "page.asp"  
decodeURIComponent(enc);               // decode a URI component  
encodeURIComponent(uri);               // encode a URI component  
parseFloat();                           // returns floating point number of string  
parseInt();                            // parses a string and returns an integer  
isFinite();                            // is variable a finite, legal number  
isNaN();                               // is variable an illegal number
```

## Events:

```
<button onclick="myFunction()"> Click here </button>
```

- **Mouse Events:**

- **onclick**
- **oncontextmenu**
- **ondblclick**
- **onmousedown**
- **onmouseenter**
- **onmouseleave**
- **onmousemove**
- **onmouseover**
- **onmouseout**
- **onmouseup**

- **Keyboard Events:**

- **onkeydown**
- **onkeypress**
- **onkeyup**

- **Frame Events:**

- **onabort**
- **onbeforeunload**
- **onerror**
- **onhashchange**
- **onload**
- **onpageshow**
- **onpagehide**
- **onresize,**
- **onscroll**
- **onunload**

- **Form Events:**

- **onblur**
- **onchange**
- **onfocus**
- **onfocusin**
- **onfocusout**
- **oninput**
- **oninvalid**
- **onreset**
- **onsearch**
- **onselect**
- **onsubmit**

- **Drag Events:**

- **ondrag**
- **ondragend**
- **ondragenter**
- **ondragleave**
- **ondragover**
- **ondragstart**
- **ondrop**

- **Clipboard Events:**

- **oncopy**
- **oncut**
- **onpaste**

- **Media Events:**

- **onabort**
- **oncanplay**
- **oncanplaythrough**
- **ondurationchange**
- **onended**
- **onerror**
- **onloadeddata**
- **onloadedmetadata**
- **onloadstart**
- **onpause**
- **onplay**
- **onplaying**
- **onprogress**
- **onratechange**
- **onseeked**
- **onseeking**
- **onstalled**
- **onsuspend**

- **ontimeupdate**
- **onvolumechange**
- **onwaiting**

- **Animation Events:**

- **Animationend**
- **Animationiteration**
- **Animationstart**

- **Miscellaneous Events:**

- **transitioned**
- **onmessage**
- **onmousewheel**
- **ononline**
- **onoffline**
- **onpopstate**
- **onshow**
- **onstorage**
- **ontoggle**
- **onwheel**
- **ontouchcancel**
- **ontouchmove**

## Errors:

```
try { // try block of code
    undefinedFunction();
}
catch(err) { // block to handle errors
    console.log(err.message);
}
```

## Throw error:

```
throw "My error message"; // throws a text
```

## Input validation:

```
const input = document.getElementById("num"); // get input element  
try {  
    const x = input.value; // get input value  
    if (x == "") throw "empty"; // error cases  
    if (isNaN(x)) throw "not a number";  
    x = Number(x);  
    if (x > 20) throw "too high";  
} catch (err) { // if there's an error  
    console.log(`Input is ${err}`); // output error  
    console.error(err); // write the error in console  
} finally {  
    console.log("Done"); // executed regardless of the try catch block  
}
```

## Error Names:

- **RangeError:** A number is "out of range".
- **ReferenceError:** An illegal reference has occurred.
- **SyntaxError:** A syntax error has occurred.
- **TypeError:** A type error has occurred.
- **URIError:** An encodeURI() error has occurred.

## JSON: JavaScript Object Notation

```
let str = '{"names":[' + // create JSON object
  '{"first":"Koushik","lastN":"Sadhu"},' +
  '{"first":"Pranay","lastN":"Gupta"},' +
  '{"first":"Shuvam","last":"Chodhury"}]};  
obj = JSON.parse(str);  
console.log(obj.names[1].first);
```

### // Send:

```
let myObj = { "name":"Nidhi", "age":20, "city":"Kolkata" }; // create object  
let myJSON = JSON.stringify(myObj); // stringify  
window.location = `demo.html?x=${myJSON}`; // sending to php
```

### // Storing and retrieving:

```
let myObj = { "name":"Nidhi", "age":20, "city":"Kolkata" };  
let myJSON = JSON.stringify(myObj); // storing data  
localStorage.setItem("testJSON", myJSON);  
let text = localStorage.getItem("testJSON"); // retrieving data  
let obj = JSON.parse(text);  
document.write(obj.name);
```

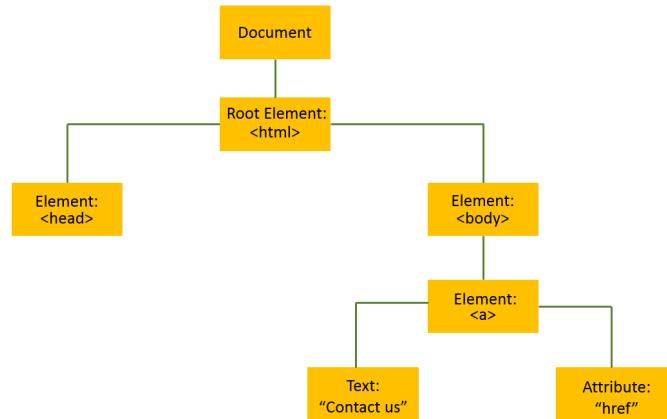
## Promises:

```
function sum(a, b) {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      // send the response after 1 second  
      if (typeof a !== "number" || typeof b !== "number") { // testing input types  
        return reject(new TypeError("Inputs must be numbers"));  
      }  
      resolve(a + b);  
    }, 1000);  
  });  
}  
  
let myPromise = sum(10, 5);  
  
myPromise.then((result) => {  
  console.log("10 + 5: ", result);  
  return sum(null, "foo"); // Invalid data and return another promise  
})  
.then(() => { // Won't be called because of the error  
})  
.catch((err) => {  
  console.error(err); // => Inputs must be numbers  
});
```

# JavaScript and the DOM

## Nodes in DOM tree

A *node* in the DOM tree is the intersection of two branches containing data. Nodes can represent HTML elements, text, attributes, etc. The *root node* is the top-most node of the tree. The illustration shows a representation of a DOM containing different types of nodes.



## HTML DOM

The DOM is an interface between scripting languages and a web page's structure. The browser creates a Document Object Model or DOM for each webpage it renders. The DOM allows scripting languages to access and modify a web page. With the help of DOM, JavaScript has the ability to create dynamic HTML.

## Accessing HTML attributes in DOM

The DOM nodes of type *Element* allow access to the same attributes available to HTML elements. For instance, for the given HTML element, the *id* attribute will be accessible through the DOM.

```
<h1 id="heading">Welcome!</h1>
```

## The Document Object Model

The *Document Object Model*, or DOM is a representation of a document (like an HTML page) as a group of objects. While it is often used to represent HTML documents, and most web browsers use JavaScript interfaces to the DOM, it is language agnostic as a model.

The DOM is tree-like and hierarchical, meaning that there is a single top-level object, and other objects descend from it in a branching structure.

## The DOM Parent-Child Relationship

The parent-child relationship observed in the DOM is reflected in the HTML nesting syntax.

Elements that are nested inside the opening and closing tag of another element are the children of that element in the DOM.

In the code block, the two `<p>` tags are children of the `<body>`, and the `<body>` is the parent of both `<p>` tags.

```
<body>
  <p>first child</p>
  <p>second child</p>
</body>
```

## The `removeChild()` Method

The `.removeChild()` method removes a specified child from a parent element. We can use this method by calling `.removeChild()` on the parent node whose child we want to remove, and passing in the child node as the argument.

In the example code block, we are removing `iceCream` from our `groceryList` element.

```
const groceryList =
document.getElementById('groceryList');
const iceCream =
document.getElementById('iceCream');

groceryList.removeChild(iceCream);
```

## The element.parentNode Property

The `.parentNode` property of an element can be used to return a reference to its direct parent node. `.parentNode` can be used on any node.

In the code block above, we are calling on the `parentNode` of the `#first-child` element to get a reference to the `#parent` `div` element.

```
<div id="parent">
  <p id ="first-child">Some child
  text</p>
  <p id ="second-child">Some more child
  text</p>
</div>
<script>
  const firstChild =
  document.getElementById('first-child');
  firstChild.parentNode; // reference to
  the #parent div
</script>
```

## The document.createElement() Method

The `document.createElement()` method creates and returns a reference to a new Element Node with the specified tag name.

`document.createElement()` does not actually add the new element to the DOM, it must be attached with a method such as `element.appendChild()`.

```
const newButton =
document.createElement("button");
```

## The element.innerHTML Property

The `element.innerHTML` property can be used to access the HTML markup that makes up an element's contents.

`element.innerHTML` can be used to access the current value of an element's contents or to reassign it.

In the code block above, we are reassigning the `box` element's inner HTML to a paragraph element with the text "Goodbye".

```
<box>
  <p>Hello there!</p>
</box>

<script>
  const box =
  document.querySelector('box');
  // Outputs '<p>Hello there!</p>':
  console.log(box.innerHTML)
  // Reassigns the value:
  box.innerHTML = '<p>Goodbye</p>'
</script>
```

## The document Object

The `document` object provides a Javascript interface to access the DOM. It can be used for a variety of purposes including referencing the `<body>` element, referencing a specific element with ID, creating new HTML elements, etc.

The given code block can be used to obtain the reference to the `<body>` element using the `document` object.

```
const body = document.body;
```

## The `document.getElementById()` Method

The `document.getElementById()` method returns the element that has the `id` attribute with the specified value.

`document.getElementById()` returns `null` if no elements with the specified ID exists.

An ID should be unique within a page. However, if more than one element with the specified ID exists, the `.getElementById()` method returns the first element in the source code.

```
// Save a reference to the element with  
// id 'demo':  
const demoElement =  
document.getElementById('demo');
```

## The `.querySelector()` Method

The `.querySelector()` method selects the first child/descendant element that matches its selector argument.

It can be invoked on the `document` object to search the entire document or on a single element instance to search that element's descendants.

In the above code block, we are using `.querySelector()` to select the first `div` element on the page, and to select the first element with a class of `button`, inside the `.main-navigation` element.

```
// Select the first <div>  
const firstDiv =  
document.querySelector('div');  
  
// Select the first .button element  
// inside .main-navigation  
const navMenu =  
document.getElementById('main-  
navigation');  
const firstButtonChild =  
navMenu.querySelector('.button');
```

## The `document.body` Object

`document.body` returns a reference to the contents of the `<body>` HTML element of a document/HTML page. The `<body>` element contains all the visible contents of the page.

## The element.onclick Property

The `element.onclick` property can be used to set a function to run when an element is clicked. For instance, the given code block will add an `<li>` element each time the element with ID `addItem` is clicked by the user.

```
let element =
document.getElementById('addItem');
element.onclick = function() {
  let newElement =
document.createElement('li');

document.getElementById('list').appendChild(newElement);
};
```

## The element.appendChild() Method

The `element.appendChild()` method appends an element as the last child of the parent.

In the given code block, a newly created `<li>` element will be appended as the last child of the HTML element with the ID `list`.

```
var node1 = document.createElement('li');
document.getElementById('list').appendChild(node1);
```

## The element.style Property

The `element.style` property can be used to access or set the CSS style rules of an element. To do so, values are assigned to the attributes of `element.style`.

In the example code, `blueElement` contains the HTML element with the ID `colorful-element`. By setting the `backgroundColor` attribute of the `style` property to blue, the CSS property `background-color` becomes blue.

Also note that, if the CSS property contains a hyphen, such as `font-family` or `background-color`, Camel Case notation is used in Javascript for the attribute name, so `background-color` becomes `backgroundColor`.

```
let blueElement =
document.getElementById('colorful-
element');
blueElement.style.backgroundColor =
'blue';
```



# **NODE JS**

# **CHEAT SHEET**

# Running Node.js

For running Node.js:

Command	Comments
node	Run the Node REPL in your terminal
node —version	Print your current Node version
node filename.js	Execute the Node code in filename.js



REPL stands for **R**ead **E**val **P**rint **L**oop. This is the list of steps that happen when you run the `node` command and then type some code.

## Node.js Global Object

In Node, we have a `global` object that we can always access. Features that we expect to be available everywhere live in this `global` object.

For example, to have some code execute after 5 seconds we can use either `global.setTimeout` or just `setTimeout`. The `global` keyword is optional.

```
setTimeout(() => {
  console.log('hello');
}, 5000);
```

Probably the most famous global is `global.console.log` which we write as just `console.log`.

## Node.js Module System

In Node.js each file is treated as a separate module. Modules provide us a way of re-using existing code.

### The `require` Function

We can re-use existing code by using the Node built-in `require()` function. This function imports code from another module.

```
const fs = require('fs');
fs.readFileSync('hello.txt');

// OR...

const { readFileSync } = require('fs');
readFileSync('hello.txt');
```

## Built-in Modules

Some modules like `fs` are built in to Node. These modules contain Node-specific features.

Key built-in modules include:

- **fs** - read and write files on your file system
- **path** - combine paths regardless of which OS you're using
- **process** - information about the currently running process, e.g. `process.argv` for arguments passed in or `process.env` for environment variables
- **http** - make requests and create HTTP servers
- **https** - work with secure HTTP servers using SSL/TLS
- **events** - work with the `EventEmitter`
- **crypto** - cryptography tools like encryption and hashing

## Creating Modules

We can create our own modules by exporting a function from a file and importing it in another module.

```
// In src/fileModule.js
function read(filename) { }
function write(filename, data) { }

module.exports = {
  read,
  write,
};

// In src/sayHello.js
```

```
const { write } = require('./fileModule.js')
write('hello.txt', 'Hello world!');
```

Some Node modules may instead use the shorthand syntax to export functions.

```
// In src/fileModule.js
exports.read = function read(filename) { }
exports.write = function write(filename, data) { }
```

## ECMAScript Modules

The imports above use a syntax known as CommonJS (CJS) modules. Node treats JavaScript code as CommonJS modules by default. More recently, you may have seen the ECMAScript module (ESM) syntax. This is the syntax that is used by TypeScript.

```
// In src/fileModule.mjs
function read(filename) { }
function write(filename, data) { }

export {
  read,
  write,
};

// In src/sayHello.mjs
import { write } from './response.mjs';
write('hello.txt', 'Hello world!');
```

We tell Node to treat JavaScript code as an ECMAScript module by using the .mjs file extension. Pick one approach and use it consistently throughout your Node project.

## Node.js Packages

Node developers often publicly share *packages*, that other developers can use to help solve common problems. A package is a collection of Node modules along with a *package.json* file describing the package.

To work with Node packages we use NPM. NPM includes two things:

1. The NPM registry with a massive collection of Node packages for us to use.

## 2. The NPM tool that you installed when you installed Node.



We can search the NPM registry for packages at [www.npmjs.com](http://www.npmjs.com). The NPM tool will by default install packages from this NPM registry.

## NPM Commands

Command	Comments
npm start	Execute the current Node package defined by package.json. Defaults to executing <code>node server.js</code> .
npm init	Initialize a fresh package.json file
npm init -y	Initialize a fresh package.json file, accepting all default options. Equivalent to npm init —yes
npm install	Equivalent to npm i
npm install <package>	Install a package from the NPM registry at <a href="http://www.npmjs.com">www.npmjs.com</a> Equivalent to npm i <package>
npm install -D <package>	Install a package as a development dependency Equivalent to npm install —save-dev <package>
npm install -g <package>	Install a package globally.
npm update <package>	Update an already installed package Equivalent to npm up <package>
npm uninstall <package>	Uninstall a package from your node_modules/ folder Equivalent to npm un <package>
npm outdated	Check for outdated package dependencies
npm audit	Check for security vulnerabilities in package dependencies
npm audit fix	Try to fix any security vulnerabilities by automatically updating vulnerable packages.

## package.json

Most Node applications we create include a `package.json` file, which means our Node applications are also Node packages.

The `package.json` file contains:

1. Name, version, description, license of the current package.
2. Scripts to automate tasks like starting, testing, and installing the current package.
3. Lists of dependencies that are required to be installed by the current package.

## node\_modules

This folder lives next to your package.json file.

When you run `npm install` the packages listed as dependencies in your package.json are downloaded from the NPM registry and put in the node\_modules folder.

It contains not just your *direct dependencies*, but also the dependencies of those dependencies. The entire *dependency tree* lives in node\_modules.

## package-lock.json

The package-lock.json file is automatically created by NPM to track the exact versions of packages that are installed in your node\_modules folder. Share your package-lock.json with other developers on your team to ensure that everyone is running the exact same versions of every package in the dependency tree.

## Node.js Event Emitter

Node.js provides a built-in module to work with events.

```
const EventEmitter = require('events');
const celebrity = new EventEmitter();

celebrity.on('success', () => {
  console.log('Congratulations! You are the best!');
});

celebrity.emit('success'); // logs success message
celebrity.emit('success'); // logs success message again
celebrity.emit('failure'); // logs nothing
```

Many features of Node are modelled with the EventEmitter class. Some examples include the currently running Node `process`, a running HTTP server, and web sockets. They all *emit* events that can then be listened for using a listener function like `on()`.

For example, we can listen for the exit event on the current running process. In this case, the event has a code associated with it to be more specific about how the process is exiting.

```
const process = require('process');

process.on('exit', (code) => {
  console.log(`About to exit with code: ${code}`);
});
```

## Backend Concepts

### ▼ Client-server architecture

Your frontend is usually the client. Your backend is usually the server.

In a client-server architecture, clients get access to data (or "resources") from the server. The client can then display and interact with this data.

The client and server communicate with each other using the HTTP protocol.

### ▼ API

Short for Application Programming Interface.

This is the set of functions or operations that your backend server supports. The frontend interacts with the backend by using only these operations.

On the web, backend APIs are commonly defined by a list of URLs, corresponding HTTP methods, and any queries and parameters.

### ▼ CRUD

Short for Create Read Update and Delete.

These are the basic operations that every API supports on collections of data. Your API will usually save (or "persist") these collections of data in a database.

### ▼ RESTful

RESTful APIs are those that follow certain constraints. These include:

- Client-server architecture. Clients get access to resources from the server using the HTTP protocol.

- Stateless communication. Each request contains all the information required by the server to handle that request. Every request is separate from every other request.
- Cacheable. The stateless communication makes caching easier.

In RESTful APIs each of our CRUD operations corresponds to an HTTP method.

CRUD Operation	HTTP method	Example
Create	POST	POST /cards Save a new card to the cards collection
Read	GET	GET /cards Get the whole cards collection or... GET /cards/:cardId Get an individual card
Update	PUT (or more rarely PATCH)	PUT /cards/:cardId Update an individual card
Delete	DELETE	DELETE /cards/:cardId Delete an individual card or more rarely... DELETE /cards Delete the entire collection of cards

# Express.js

## GET Routes

```
// Get a whole collection of JSON objects
app.get("/cards", (req, res) => {
  return res.json(cards);
});

// Get a specific item in a collection by ID
app.get("/cards/:cardId", (req, res) => {
  const cardId = req.params.cardId;
  return res.json(cards[cardId]);
});
```

## POST Routes

```
app.post("/cards", (req, res) => {
  // Get body from the request
  const card = req.body;

  // Validate the body
```

```

if (!card.value || !card.suit) {
  return res.status(400).json({
    error: 'Missing required card property',
  });
}

// Update your collection
cards.push(card);

// Send saved object in the response to verify
return res.json(card);
});

```

## Routers

```

// In src/cards.router.js
const cardsRouter = express.Router();

cardsRouter.get("/", (req, res) => {
  return res.json(cards);
});

cardsRouter.get("/:cardId", (req, res) => {
  const cardId = req.params.cardId;
  return res.json(cards[cardId]);
});

// In src/api.js
const cardsRouter = require('./cards.router');

const api = express.Router();

api.use('/cards', cardsRouter);

```

## Node.js Folder Structure

One typical folder structure for an API following RESTful architecture and using the Express framework can be found below. Node servers typically follow the Model View Controller pattern. Models live together in one folder. Controllers are grouped together based on which feature or collection they are related to. Views are typically managed by the front end, although some Node servers may serve static HTML or use templating engines like [Handlebars](#).

```

node-project/                      # top level project
  node_modules/                   # all installed node packages
  data/                          # static data files, if needed
    database.json
  src/                           # the source code for your server
    models/                       # models following the model-view-controller pattern
      comment.model.js
      post.model.js
    routes/                        # one folder for each collection in your API
      feeds/                         # folder for the user feeds collection
        feed.router.js
        feed.controller.js
      posts/                         # router listing all possible routes for user feeds
        post.router.js
        post.controller.js
    api.js                          # controller with the implementation for each route
  services/                        # top level router connecting all the above routes
    mongo.js                        # any long running services or utilities
    app.js                          # e.g. connecting to a MongoDB database
    server.js                       # all Express middleware and routers
    .gitignore
  package-lock.json
  package.json

```

This is just a reference. In the real world, every project will have differences in the requirements and the ideal project structure.

## Cross Origin Resource Sharing

Something *all* web developers soon come across is **Cross Origin Resource Sharing (CORS)**.

Browsers follow the **Same Origin Policy (SOP)**, which prevents requests being made across different origins. This is designed to stop malicious servers from stealing information that doesn't belong to them.

**Cross Origin Resource Sharing (CORS)** allows us to allow or whitelist other origins that we trust, so that we can make requests to servers that don't belong to us. For example, with CORS set up properly, <https://www.mydomain.com> could make a POST request to <https://www.yourdomain.com>

In Express we commonly set up CORS using the following middleware package:  
<https://www.npmjs.com/package/cors>

# PM2 Commands

PM2 is a tool we use to create and manage Node.js clusters. It allows us to create clusters of processes, to manage those processes in production, and to keep our applications running forever. We can install the PM2 tool globally using `npm install -g pm2`.

Command	Comments
pm2 list	List the status of all processes managed by PM2.
pm2 start server.js -i 4	Start server.js in cluster mode with 4 processes.
pm2 start server.js -i 0	Start server.js in cluster mode with the maximum number of processes to take full advantage of your CPU.
pm2 logs	Show logs from all processes.
pm2 logs —lines 200	Show older logs up to 200 lines long.
pm2 monit	Display a real-time dashboard in your terminal with statistics for all processes.
pm2 stop 0	Stop running process with ID 0.
pm2 restart 0	Restart process with ID 0.
pm2 delete 0	Remove process with ID 0 from PM2's list of managed processes.
pm2 delete all	Remove all processes from PM2's list.
pm2 reload all	Zero downtime reload of all processes managed by PM2. For updating and reloading server code already running in production.



# Express

## CheatSheet



In this Cheatsheet, we will cover the basics of Express.js. We will provide examples to help you understand how Express.js works and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.



Express.js is a popular open-source web application framework for Node.js, a JavaScript runtime that allows developers to build scalable, high-performance server-side applications. It provides a wide range of features and utilities for web application development, such as routing, middleware, and template rendering.

With Express.js, developers can quickly create server-side applications that handle HTTP requests, process data, and respond with dynamic content or data from a database. It also supports a wide range of middleware that can be used to add functionality, such as authentication, compression, logging, and more.

Express.js is designed to be flexible and easy to use, allowing developers to customize and extend it to meet the specific needs of their applications. Its modular structure and extensive community support make it one of the most popular frameworks for building web applications with Node.js.

**Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.**

# INSTALLATION

```
npm install express
```

## HELLO WORLD SERVER

```
const express = require('express');
const app = express();
app.set('port', (process.env.PORT || config.port));
app.get('/', (req, res) => res.send('Hello World!'));
app.listen(app.get('port'), () =>
  console.log(`Server started on ${app.get('port')} port`))
```

## BASIC ROUTING:

```
// GET
app.get('/', function (req, res) {
  res.send('Hello World!')
})

// POST
app.post('/', function (req, res) {
  res.send('POST request. body:', req.body)
})

// DELETE
app.delete('/:id', function (req, res) {
  res.send('DELETE request for id:'. req.params.id)
})
```

## STATIC FILE SERVING:

```
app.use(express.static(__dirname + '/public'));
```

## LOGGING ALL ROUTES:

```
app._router.stack.forEach(function(r) {
  if (r.route && r.route.path) {
    console.log(r.route.path)
  }
});
```

## DEFINING ROUTES IN A DIFFERENT FILE:

File: /routes/users.js

```
// File Path: /routes/users.js

var express = require('express');
var router = express.Router();
router.get('/', (req, res) => {
  const users = []; // get from db
  res.send(users);
});
router.get('/:id', (req, res) => {
  const user = {}; // get from db
  res.send(user);
});
router.post('/', (req, res) => {
  const user = req.body; // save user to db
  res.send({status: 'success'});
});
module.exports = router;
```

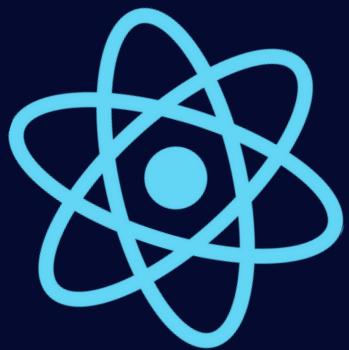
## ADDING ROUTES FROM: /routes/users.js

```
app.use('/user', require('./routes/user'));
```

## REDIRECTS:

```
router.get('/old-path', function(req, res) {
  res.redirect('/temp-new-path'); // sends a 302
});

router.get('/old-path', function(req, res) {
  res.redirect(301, '/permanent-new-path'); // sends a 301
});
```



# React

## CheatSheet



In this Cheatsheet, we will cover the basics of React. We will provide examples to help you understand how React works and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.



React is a JavaScript library for building user interfaces. It was developed and is maintained by Facebook and a community of individual developers and companies. React allows developers to build reusable UI components, manage the state of their applications, and render dynamic updates efficiently.

React uses a virtual DOM (Document Object Model) to update the view without having to reload the entire page. This results in fast and smooth updates to the user interface. React also uses a component-based architecture, which allows developers to easily reuse and compose their components to build complex UIs.

React is used for both web and mobile development, and is popular for its simplicity, versatility, and performance. Many companies and organizations use React for their websites and applications, including Facebook, Airbnb, Netflix, and Dropbox.

# MANAGE STATE

## useState

```
const [count, setCount] = useState(initialCount);
```

- Class way

```
const initialCount = 0;
class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = { count: initialCount };
  }
  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button
          onClick = {() => this.setState(({count}) => ({ count: count + 1 }))}
        >
          Click me
        </button>
      </div>
    );
  }
}
```

- Hook Way

```
import { useState } from "react";
const initialCount = 0;
function Counter() {
  const [count, setCount] = useState(initialCount);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button
        onClick={() => setCount((c) => c + 1)}
      >
        Click me
      </button>
    </div>
  );
}
```

## useReducer

```
const [state, dispatch] = useReducer(
  reducer,
  initialState,
  initialDispatch
);
```

An alternative to useState. Use it in the components that need complex state management, such as multiple state values being updated by multiple methods.

```
const initialState = {count: 0};
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}
function Counter({initialState}) {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
    Count: {state.count}
    <button onClick={() => dispatch({type: 'increment'})}>+</button>
    <button onClick={() => dispatch({type: 'decrement'})}>-</button>
    </>
  );
}
```

## HANDLE SIDE EFFECTS

### useEffect

```
useEffect(() => {
  applyEffect(dependencies);
  return () => cleanupEffect();
}, [dependencies]);
```

- Class way

```
class FriendStatus extends React.Component {  
    state = { isOnline: null };  
    componentDidMount() {  
        ChatAPI.subscribeToFriendStatus(  
            this.props.friend.id,  
            this.handleStatusChange  
        );  
    }  
    componentWillUnmount() {  
        ChatAPI.unsubscribeFromFriendStatus(  
            this.props.friend.id,  
            this.handleStatusChange  
        );  
    }  
    componentDidUpdate(prevProps) {  
        if(this.props.friend.id !== prevProps.id) {  
            ChatAPI.unsubscribeFromFriendStatus(  
                prevProps.friend.id,  
                this.handleStatusChange  
            );  
            ChatAPI.subscribeToFriendStatus(  
                this.props.friend.id,  
                this.handleStatusChange  
            );  
        }  
    }  
    handleStatusChange = status => {  
        this.setState({  
            isOnline: status.isOnline  
        });  
    }  
    render() {  
        if (this.state.isOnline === null) {  
            return 'Loading...';  
        }  
        return this.state.isOnline ? 'Online' : 'Offline';  
    }  
}
```

- Hook Way

```
import { useState, useEffect } from 'react';
function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);
  status
  const handleStatusChange = (status) => setIsOnline(status.isOnline);
  useEffect(
    () => {
      // Update status when the listener triggers
      ChatAPI.subscribeToFriendStatus(
        props.friend.id,
        handleStatusChange
      );
      // Stop listening to status changes every time we cleanup
      return function cleanup() {
        ChatAPI.unsubscribeFromFriendStatus(
          props.friend.id,
          handleStatusChange
        );
      };
    },
    [props.friend.id] // Cleanup when friend id changes or when we "unmount"
  );
  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

## useLayoutEffect

```
useLayoutEffect(() => {
  applyBlockingEffect(dependencies);
  return cleanupEffect();
}, [dependencies]);
```

`useLayoutEffect` is almost same as `useEffect`, but fires synchronously after the render phase. Use this to safely read from or write to the DOM

```
import { useRef, useLayoutEffect } from "react";
function ColoredComponent({color}) {
  const ref = useRef();
  useLayoutEffect(() => {
    const refColor = ref.current.style.color;
    console.log(` ${refColor} will always be the same as ${color}`);
    ref.current.style.color = "rgba(255,0,0)";
  }, [color]);
  useEffect(() => {
    const refColor = ref.current.style.color;
    console.log(
      `but ${refColor} can be different from ${color} if you play with the
DOM` );
  }, [color]);
  return (
    <div ref={ref} style={{ color: color }}>
      Hello React hooks !
    </div>
  );
}
```

```
<ColoredComponent color = {"rgb(42, 13, 37)"} />
// rgb(42, 13, 37) will always be the same as rgb(42, 13, 37)
// but rgba(255, 0, 0) can be different from rgb(42, 13, 37) if you play with the DOM
```

# USE THE CONTEXT API

## useContext

```
const ThemeContext = React.createContext();
const contextValue = useContext(ThemeContext);
```

- Class way

```
class Header extends React.Component {
  public render() {
    return (
      <AuthContext.Consumer>
        {({ handleLogin, isLoggedIn }) => (
          <ModalContext.Consumer>
            {({ isOpen, showModal, hideModal }) => (
              <NotificationContext.Consumer>
                {({ notification, notify }) => {
                  return (
                    ...
                  )
                }
              </NotificationContext.Consumer>
            )}
          </ModalContext.Consumer>
        )}
      </AuthContext.Consumer>
    );
  }
}
```

- Hook Way

```
import { useContext } from 'react';
function Header() {
  const { handleLogin, isLoggedIn } = useContext(AuthContext);
  const { isOpen, showModal, hideModal } = useContext(ModalContext);
  const { notification, notify } = useContext(NotificationContext);

  return ...
}
```

**NOTE:** Use the created context, not the consumer.

```
const Context = React.createContext(defaultValue);
// Wrong
const value = useContext(Context.Consumer);
// Right
const value = useContext(Context);
```

## MEMOIZE EVERYTHING

### useMemo

```
const memoizedValue = useMemo (
  () => expensiveFn(dependencies),
  [dependencies]
);
```

```

function RandomColoredLetter(props) {
  const [color, setColor] = useState('#fff')
  const [letter, setLetter] = useState('a')
  const handleColorChange = useMemo(() => () =>
  setColor(randomColor()), []);
  const handleLetterChange = useMemo(() => () =>
  setLetter(randomColor()), []);
  return (
    <div>
      <ColorPicker handleChange={handleColorChange} color={color} />
      <LetterPicker handleChange={handleLetterChange} letter={letter} />
      <hr/>
      <h1 style={{color}}>{letter}</h1>
    </div>
  )
}

```

## useCallback

```

const memoizedCallback = useCallback (
  expensiveFn (dependencies),
  [dependencies]
);

```

```

function RandomColoredLetter(props) {
  const [color, setColor] = useState('#fff')
  const [letter, setLetter] = useState('a')
  const handleColorChange = useCallback(() => setColor(randomColor()), []);
  const handleLetterChange = useCallback(() => setLetter(randomColor()), []);
  return (
    <div>
      <ColorPicker handleChange={handleColorChange} color={color} />
      <LetterPicker handleChange={handleLetterChange} letter={letter} />
      <hr/>
      <h1 style={{color}}>{letter}</h1>
    </div>
  )
}

```

# USE REFS

## useRef

```
const ref = useRef();
```

**useRef** can just be used as a common React ref :

```
import { useRef } from "react";
function TextInput() {
  const inputRef = useRef(null);
  const onBtnClick = () => inputRef.current.focus();
  return (
    <>
    <input ref={inputRef} />
    <button onClick={onBtnClick}>Focus the text input</button>
    </>
  )
}
```

But it also allows you to just hold a mutable value through any render. Also, mutating the value of `ref.current` will not cause any render.

## useImperativeHandle

```
useImperativeHandle (
  ref,
  createHandle,
  [dependencies]
)
```

`useImperativeHandle` allows you to customize the exposed interface of a component when using a ref. The following component will automatically focus the child input when mounted :

```
function TextInput(props, ref) {
  const inputRef = useRef(null);
  const onBtnClick = () => inputRef.current.focus();
  useImperativeHandle(ref, () => ({
    focusInput: () => inputRef.current.focus();
  });
  return (
    <Fragment>
      <input ref={inputRef} />
      <button onClick={onBtnClick}>Focus the text input</button>
    </Fragment>
  )
}
const TextInputWithRef = React.forwardRef(TextInput);
function Parent() {
  const ref = useRef(null);
  useEffect(() => {
    ref.focusInput();
  }, []);
  return (
    <div>
      <TextInputWithRef ref={ref} />
    </div>
  );
}
```

# REUSABILITY

Extract reusable behaviour into custom hooks.

```
import { useState, useRef, useCallback, useEffect } from "React";
// let's hide the complexity of listening to hover changes
function useHover() {
    const [value, setValue] = useState(false); // store the hovered state
    const ref = useRef(null); // expose a ref to listen to
    // memoize function calls
    const handleMouseOver = useCallback(() => setValue(true), []);
    const handleMouseOut = useCallback(() => setValue(false), []);
    // add listeners inside an effect,
    // and listen for ref changes to apply the effect again
    useEffect(() => {
        const node = ref.current;
        if (node) {
            node.addEventListener("mouseover", handleMouseOver);
            node.addEventListener("mouseout", handleMouseOut);
        }
        return () => {
            node.removeEventListener("mouseover", handleMouseOver);
            node.removeEventListener("mouseout", handleMouseOut);
        };
    }, [ref.current]);
    // return the pair of the exposed ref and it's hovered state
    return [ref, value];
}
```

```
const HoverableComponent = () => {
    const [ref, isHovered] = useHover();
    return (
        <span style={{ color: isHovered ? "blue" : "red" }} ref={ref}>
            Hello React hooks !
        </span>
    );
};
```



# APIs

## CheatSheet



In this Cheatsheet, we will cover the basics of API. We will provide examples to help you understand how API's work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.



An API, or Application Programming Interface, is a set of rules and protocols for building and interacting with software applications. It allows different software systems to communicate with each other, enabling them to share data and functionality. This allows developers to access the functionality of a certain software application or system without having to understand its underlying code or implementation.

An example of an API is the Facebook API, which allows developers to access and interact with the functionality of the Facebook platform, such as posting status updates, retrieving user information, and managing ad campaigns. Another example is the Google Maps API, which allows developers to embed maps and location-based functionality in their own websites and apps.

### **How an API Works:**

APIs act as a bridge between applications and web servers, processing data transfer between systems. When a client application initiates an API call, also known as a request, it is sent to the web server via the API's Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes a request body. The API then processes the request and may make a call to an external program or web server for the requested information.

The server responds with the requested data, which the API then forwards to the initial requesting application. This process of requests and responses all happens through the API. Unlike user interfaces which are designed for human use, APIs are designed for use by computers or applications.

## **REST API: (Representational State Transfer)**

REST (Representational State Transfer) is a type of web architecture and a set of constraints to be used when creating web services. RESTful API (Application Programming Interface) is an API that conforms to the REST architectural style and constraints, and it is typically used to make requests to retrieve or update data on a web server. A RESTful API uses HTTP requests to POST (create), PUT (update), GET (read), and DELETE (delete) data. A RESTful API also returns a response in a standard format, typically JSON or XML, and uses standard HTTP status codes to indicate the status of the request. RESTful APIs are popular because they are simple to understand and easy to use, and they work well with the HTTP protocol that the internet is built on. Additionally, RESTful APIs are often faster and more lightweight than their SOAP (Simple Object Access Protocol) counterparts because they use smaller message formats. RESTful API's have become a popular way for systems to expose databases through HTTP(S) following CRUD operations (Create, Read, Update, Delete), and return JSON or XML as responses, it's also widely used in microservices, mobile and web applications, IoT, and many more.

REST requires that a client make a request to the server in order to retrieve or modify data on the server.

## A request generally consists:

- An HTTP verb, which defines what kind of operation to perform.
- A header, which allows the client to pass along information about the request.
- A path to a resource.
- An optional message body containing data.

## CRUD: (Create Read Update Delete)

CRUD stands for Create, Read, Update, and Delete, which are the four basic operations that can be performed on data in a database. These operations are often used to interact with databases through APIs (Application Programming Interfaces).

- **Create:** This operation is used to add new data to the database. It is typically performed using the **HTTP POST** method and is used to create new resources.
- **Read:** This operation is used to retrieve data from the database. It is typically performed using the **HTTP GET** method and is used to read existing resources.
- **Update:** This operation is used to modify existing data in the database. It is typically performed using the **HTTP PUT** method and is used to update existing resources.
- **Delete:** This operation is used to remove data from the database. It is typically performed using the **HTTP DELETE** method and is used to delete resources.

CRUD operations are typically implemented in RESTful APIs, but they can also be used in other types of APIs. These operations can be used to expose the data in a database to other systems, such as a mobile app or a web application, that can use the data to provide a service to end-users.

## HTTP Verbs

In the context of API (Application Programming Interface), HTTP verbs (or methods) are used to specify the type of action that the API client wants to perform on a resource. The most commonly used HTTP verbs in API are:

- **GET:** This verb is used to retrieve information from the server. It is the most common verb used to make a request to an API. It is considered safe, meaning that it should not have any side effects on the server or the data it serves.
- **POST:** This verb is used to submit information to the server. It is typically used to create new resources.
- **DELETE:** This verb is used to delete resources.
- **PUT:** This verb is used to update existing resources. It replaces the entire resource, unlike the PATCH verb which modifies only the fields sent in the request.
- **PATCH:** This verb is used to partially update a resource. It is used to modify only the fields sent in the request and it's usually used when you don't want to replace all the resource's attributes.

- **PUT vs PATCH:** PUT method uses the request URI to supply a modified version of the requested resource which replaces the original version of the resource, whereas the PATCH method supplies a set of instructions to modify the resource.
- **HEAD:** This verb is similar to GET, but it only returns the headers of the response, without the body.
- **OPTIONS:** This verb is used to retrieve the allowed actions on a resource.
- **CONNECT:** Connect request establishes a tunnel to the server identified by a specific URI. A good example is SSL tunnelling.
- **TRACE:** The Trace method performs a message loop-back test along the path to the target resource, to provide a useful debugging mechanism. It allows clients to view whatever message is being received at the other end of the request chain so that they can use the info for testing or diagnostic functions.

These verbs are part of the HTTP protocol, and are commonly used in RESTful (Representational State Transfer) APIs, which are built on top of the HTTP protocol and are designed to be easily consumed by web and mobile applications.

# HTTP Status Codes

HTTP status codes are three-digit numbers returned by an API (Application Programming Interface) to indicate the status of a request. These codes provide information about whether a request was successful or not, and they are an important part of the API development.

The first digit of the status code specifies one of five standard classes of responses.

## Standard Classes:

- **1xx:** Informational responses.
- **2xx:** Successful responses.
- **3xx:** Redirection responses.
- **4xx:** Client error responses.
- **5xx:** Server error responses.

### **1xx: Informational responses**

It indicates that the request was received and understood by the server and it's continuing the process.

- **100:** Continue.
- **101:** Switching Protocols.
- **102:** Processing.
- **103:** Early Hints.

## **2xx: Successful responses**

It indicates that the action requested by the client was received, understood, and accepted.

- **200:** OK.
- **201:** Created.
- **202:** Accepted.
- **203:** Non-Authoritative Information.
- **204:** No Content.

## **3xx: Redirection responses**

Many of these 3xx status codes are used in URL redirection or it indicates the client must take additional action to complete the request.

- **301:** Moved Permanently.
- **302:** Found.
- **304:** Not Modified.
- **305:** Use Proxy.
- **307:** Temporary Redirect.
- **308:** Permanent Redirect.

## **4xx: Client error responses**

This status code is intended for situations in which the error seems to have been caused by the client.

- **400:** Bad Request.
- **401:** Unauthorized.
- **403:** Forbidden.
- **404:** Not Found.
- **406:** Not Acceptable.
- **408:** Request Timeout.

## **5xx: Server error responses**

It indicates that the server has encountered a situation where it doesn't know how to handle a request.

- **500:** Internal Server Error
- **501:** Not Implemented
- **502:** Bad Gateway
- **503:** Service Unavailable
- **504:** Gateway Timeout
- **505:** HTTP Version Not Supported

These are just a few examples of the many HTTP status codes that can be returned by an API. It's important to understand the meaning of each code and to handle them appropriately in the client application. The HTTP status codes are standardized and are used as a way for the server to communicate with the client about the outcome of a request.



# mongo DB

## CheatSheet



In this Cheatsheet, we will cover the basics of mongo DB. We will provide examples to help you understand how mongo DB work and how to use them in your own web development projects. Whether you are a beginner or an experienced developer, this PDF can serve as a useful reference guide.



MongoDB is a document-oriented, open-source database program that is used for storing and retrieving data. It is a NoSQL database, which means that it does not use a fixed schema and does not rely on tables to organize data, unlike a traditional relational database. Instead, MongoDB stores data in a format called BSON (binary JSON), which allows for more flexibility and scalability when working with large amounts of data. MongoDB is often used for web and mobile applications, real-time analytics, and big data processing.

### Most Commonly Used MongoDB terms:

Common DB Terms	MongoDB Terms
Database	Database
Tables	Collections
Rows	Document
Columns	Fields

**NOTE:** In MongoDB, data are store in **BSON** not in **JSON** format.

# ATOMIC/UPDATE OPERATOR

Atomic Operator	Use of the Operator
<b>\$set</b>	Updating the stored document without changing the prev. state of the document.
<b>\$inc</b>	To increment and decrement on the numerical type variable.
<b>\$push</b>	Push new field but it must be of ARRAY type field.
<b>\$eq</b>	Matches values that are equal to the given value.
<b>\$gt</b>	Matches if values are greater than the given value.
<b>\$lt</b>	Matches if values are less than the given value.
<b>\$gte</b>	Matches if values are greater or equal to the given value.
<b>\$lte</b>	Matches if values are less or equal to the given value.
<b>\$in</b>	Matches any of the values in an array.
<b>\$ne</b>	Matches values that are not equal to the given value.
<b>\$nin</b>	Matches none of the values specified in an array.

## MongoDB COMMANDS

- For invoking Mongo Daemon:

```
mongod
```

- For starting the mongo shell:

```
mongo
```

- To show all the database present in the MongoDB:

```
show dbs
```

- To create a new database:

```
use students
```

- To see the current database:

```
db
```

- To delete the current database which we are in:

```
db.dropDatabase()
```

- To show all the collection in the database:

```
show collections
```

- To show all the collection in the database:

```
show collections
```

- To create a new collections:

```
db.createCollection('studentData')
```

- To delete a collection:

```
db.studentData.drop()
```

- To insert a document in the collections:

```
db.studentData.insert({name : "Jack", age : 20})
```

- For inserting only one document into the collections (same work as “ .insert() ”):

```
db.studentData.insertOne({  
    name : "Travour",  
    email : "travor458@gmail.com",  
    age : 24  
})
```

- For inserting multiple document into the collections:

```
db.studentData.insertMany([  
    {name : "Harry", email : "marry234@gmail.com", age : 13},  
    {name : "Ronit", email : "ronit@outlook.com", age : 45},  
    {name : "Adesh", email : "sadhuadesh@gmail.com", age : 18}  
])
```

- For displaying the documents present inside the collections:

```
db.studentData.find()
```

- For displaying the documents present inside the collections in a prettyer format:

```
db.studentData.find().pretty()
```

- For displaying the limited number of document present in the collection:

```
db.studentData.find().limit(3).pretty()
```

- For counting the number of document present in the collections:

```
db.studentData.find().count()
```

- For sorting the document in ascending order on the basis of age:

```
db.studentData.find().sort({age : 1})
```

- For sorting the document in descending order on the basis of age:

```
db.studentData.find().sort({age : -1})
```

- For chaining multiple function together:

```
db.studentData.find().limit(2).sort({age : -1}).pretty()
```

- For finding only one document:

```
db.studentData.findOne({age : 18})
```

- For finding a group of documents based on some condition:

```
db.studentData.findOne({age : {$lte : 18}})
```

- For updating a data based upon certain condition or filter:  
(This will change the entire documents and fields by removing the previous state of the document)

```
db.studentData.update({name : "Harry"} , {fblogged : "Yes" , age : 45})
```

- For updating a data based upon certain condition or filter:  
(This will change the documents and fields without changing the previous state of the document)

```
db.studentData.updateOne({name : "Harry"} , {$set : {fblogged : "Yes" , age : 45}})
```

- For updating a group of document based on certain condition:

```
db.studentData.updateMany({age : 24} , {$set : {courseCount : 2}})
```

- For adding a new field to all the documents:

```
db.studentData.updateMany({} , {$set : {registered : "Yes"}})
```

- For renaming a field name:

```
db.studentData.updateMany({} , {$rename : {age : "student_age"}})
```

- For incrementing a numeric field in a document:

```
db.studentData.updateMany({} , {$inc : {age : 1}})
```

- For decrementing a numeric field in a document:

```
db.studentData.updateMany(  
    {age : {$lt : 19}} ,  
    {  
        $inc : {courseCount : -1},  
        $set : {hobby : ["Painting", "Football"]}  
    })
```

- To append new data to the array type field:

```
db.studentData.updateMany(  
  {age : {$lt : 19}} ,  
  {  
    $push : {hobby : "Swimming"}  
})
```

- To delete One document using condition or filter:

```
db.studentData.deleteOne({age : 17})
```

- To delete the first document from the table:

```
db.studentData.deleteOne({})
```

- To delete the first document from the table:

```
db.studentData.deleteMany({fblogged : "Yes"})
```

- To delete the all the documents of the table:

```
db.studentData.deleteMany({})
```

- To group specific field together:  
(This will display the email and \_id)

```
db.studentData.find({}, {email : 1})
```

- To group specific field together:  
(\_id by default is 1)

```
db.studentData.find({name : "Harry"}, {email : 1, _id : 0, name : 1})
```

- Converting the group of data into an Array:

```
db.studentData.find({name : "Harry"}, {email : 1, _id : 0}).toArray()
```

- To get a particular data from a document:

```
db.studentData.findOne({name : "Adesh"}).email  
db.studentData.findOne({name : "Adesh"}).age  
db.studentData.findOne({name : "Adesh"}).hobby[1]
```

- How to store a data into a variable:

```
var studentid = db.studentData.findOne({name : "Travour"})._id  
studentid
```

- How to fetch data from the variable (studentid):

```
db.studentData.find({_id : studentid})  
db.studentData.findOne({_id : studentid}).email
```

- How to print all the documents using ForEach Loop

(If in your collection there is more than 20 documents then ‘.find()’ function will display only the first 20 documents and to display the next 20 documents you have type ‘it’ to display more So, by using for each loop you will see all the documents present in the collections.)

```
db.studentData.find().forEach((student) => {printjson(student)})
```

- How to print specific field from all the documents:

```
db.studentData.find().forEach((student) =>
```



# GitHub CheatSheet



# **GIT**

Git is a version control system that is used for tracking changes in computer files and coordinating work on those files among multiple people. It is a distributed version control system, which means that it allows multiple users to work on the same files simultaneously and keeps track of changes made to the files by each user. Git is widely used in software development and has become a standard tool for collaborating on code.

**This cheat sheet features the most important and commonly used Git commands for easy reference.**

## **INSTALLATION & GUI'S**

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

**GitHub for Windows:** <https://windows.github.com>

**GitHub for Mac:** <https://mac.github.com>

For Linux and Solaris platforms, the latest release is available on the official Git web site.

**Git for All Platforms:** <https://git-scm.com>

# **SETUP**

Configuring user information used across all local repositories.

**git config --global user.name “[firstname lastname]”**

set a name that is identifiable for credit when review version history.

**git config --global user.email “[valid-email]”**

set an email address that will be associated with each history marker.

**git config --global color.ui auto**

set automatic command line coloring for Git for easy reviewing.

# **SETUP & INIT**

Configuring user information, initializing and cloning repositories.

**git init**

initialize an existing directory as a Git repository.

**git clone [URL]**

retrieve an entire repository from a hosted location via URL.

# **STAGE & SNAPSHOT**

Working with snapshots and the Git staging area.

**git status**

show modified files in working directory, staged for your next commit.

## **git add [file]**

add a file as it looks now to your next commit (stage).

## **git reset [file]**

unstage a file while retaining the changes in working directory.

## **git diff**

diff of what is changed but not staged.

## **git diff --staged**

diff of what is staged but not yet committed.

## **git commit -m “[descriptive message]”**

commit your staged content as a new commit snapshot.

# **BRANCH & MERGE**

Isolating work in branches, changing context, and integrating changes.

## **git branch**

list your branches, a\* will appear next to the currently active branch.

## **git branch [branch-name]**

create a new branch at the current commit.

## **git checkout**

switch to another branch and check it out into your working directory.

## **git merge [branch]**

merge the specified branch's history into the current one.

## **git log**

show all commits in the current branch's history.

# **INSPECT & COMPARE**

Examining logs, diffs and object information.

## **git log**

show the commit history for the currently active branch.

## **git log branchB..branchA**

show the commits on branchA that are not on branchB.

## **git log --follow [file]**

show the commits that changed file, even across renames.

## **git diff branchB...branchA**

show the diff of what is in branchA that is not in branchB.

## **git show [SHA]**

show any object in Git in human-readable format.

# **TRACKING PATH CHANGES**

Versioning file removes and path changes.

## **git rm [file]**

delete the file from project and stage the removal for commit.

## **git mv [existing-path] [new-path]**

change an existing file path and stage the move.

## **git log --stat -M**

show all commit logs with indication of any paths that moved.

# IGNORING PATTERNS

Preventing unintentional staging or committing of files.

**logs/**  
**\*.notes**  
**pattern\*/**

Save a file with desired patterns as `.gitignore` with either direct string matches or wildcard globs.

**git config --global core.excludesfile [file]**  
system wide ignore pattern for all local repositories.

## SHARE & UPDATE

Retrieving updates from another repository and updating local repository.

**git remote add [alias] [URL]**  
add a git URL as an alias.

**git fetch [alias]**  
fetch down all the branches from that Git remote.

**git merge [alias]/[branch]**  
merge a remote branch into your current branch to bring it up to date.

**git push [alias] [branch]**  
Transmit local branch commits to the remote repository branch.

**git pull**  
fetch and merge any commits from the tracking remote branch.

# REWRITE HISTORY

Rewriting branches, updating commits and clearing history.

## **git rebase [branch]**

apply any commits of current branch ahead of specified one.

## **git reset --hard [commit]**

clear staging area, rewrite working tree from specified commit.

# TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches.

## **git stash**

Save modified and staged changes.

## **git stash list**

list stack-order of stashed file changes.

## **git stash pop**

write working from top of stash stack.

## **git stash drop**

discard the changes from top of stash stack.