
# HTML & CSS: LEVEL 1

Instructor: Beck Johnson

Week 3

# SESSION OVERVIEW

- Review of week 2
- The CSS box model
- Block vs inline elements
- A couple new HTML elements
- Understanding classes and IDs

# REVIEW!

# REVIEW: WEB GRAPHICS

- **Minimize** file sizes to help load times in browser
- **Optimizes** images for RGB displays with correct **resolution** for browsers
- **Flattens** layers and removes metadata from graphics

# REVIEW: WEB IMAGE TYPES


## JPG/JPEG

- Millions of colors
- Uses a compression algorithm called **lossy**
- No animation
- No transparency
- Small file size

# REVIEW: WEB IMAGE TYPES


## GIF

- 256 colors max
- Animation
- Pixels are either on or off (no partial transparency)

# REVIEW: WEB IMAGE TYPES

## PNG

- Millions of colors
- No animation
- Full alpha transparency
- No compression, so larger file sizes

# REVIEW: LINKING TO EXTERNAL STYLESHEET

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

- Tells the browser to find and load the styles.css file from the css directory
- The `rel` attribute stands for "relation" - in this case, this link's relationship to the document is "stylesheet"
- This tag goes inside the `<head>` element
- Should be on every page that needs the styles

# REVIEW: THE "CASCADING" PART

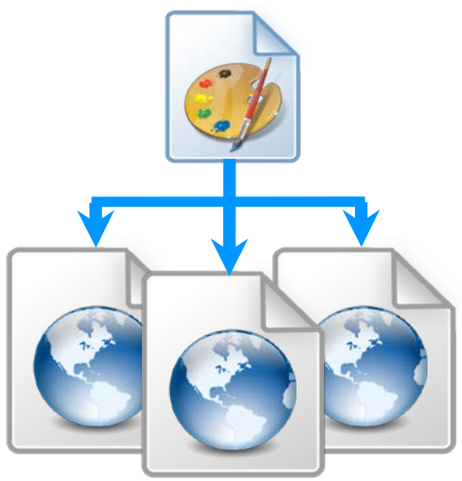
## The 3 rules for determining how styles get applied:

- Styles are applied from **far** to **near**
- Styles are applied from **top** to **bottom**
- **Children** elements are more specific than **parents**

# {} REVIEW: NEAR TO FAR

Styles that are “closer” to the elements they style take precedence

Browser default



External styles
(in a **.css** file)

Internal styles
(in the <head>)

Inline styles
(directly on an element)

Closer to element

# {} REVIEW: TOP TO BOTTOM

If the same `property` is styled multiple times for the same `selector`, the last one sticks.

```
p { color: #2f4251; }
p { color:  #daa645; } /*this wins*/
```

# {} REVIEW: CHILDREN ARE SPECIFIC

Children elements **inherit** styles from their parents but can **override** parents with their own  styles

```
body { color: #2f4251; } /* parent */
p { color: #daa645; }    /* child */
```

# {} REVIEW: SELECTORS CAN BE MORE SPECIFIC

If one style is **more specific** than another, it takes precedence

```
p { color: #daa645; } /* all paragraphs */

a { color: #e7c0c8; } /* links in general */

p a { color: #c4fe46; } /* links in paragraphs */
```

# BACKGROUND-IMAGE: REVIEW

Can set background of an element as an **image** using `background-image`

```
p {
    background-image: url("images/kitten.jpg");
    color: white;
}
```

# BACKGROUND: REVIEW

`background-position:` allows you to move a background image around within its container

`background-repeat:` defines if (and how) the background image will repeat

`background-attachment:` changes if the image stays in place when the user scrolls the page or scrolls with the page

`background-size:` specifies how much of the container that the image covers

# PSEUDO REVIEW

A **CSS pseudo-class selector** specifies a special state of the element we want to style

`:first-letter` styles the first letter of a block of text

`:first-child` and `:last-child` style the first and last children of a parent

`:focus` styles an element that has the current keyboard focus, from either click or tab

# HEIGHT AND WIDTH: REVIEW

`height` and `width` can be set on (most) elements to change how much room they take up on the page.

```
header { height: 6em; }
```

To ensure an element is **never larger** than a certain value, use `max-height` or `max-width`.

Specify `min-height` or `min-width` if you want to ensure an element is **never smaller** than a certain value.

# QUESTIONS?

# THE CSS BOX MODEL

# CSS BOX MODEL

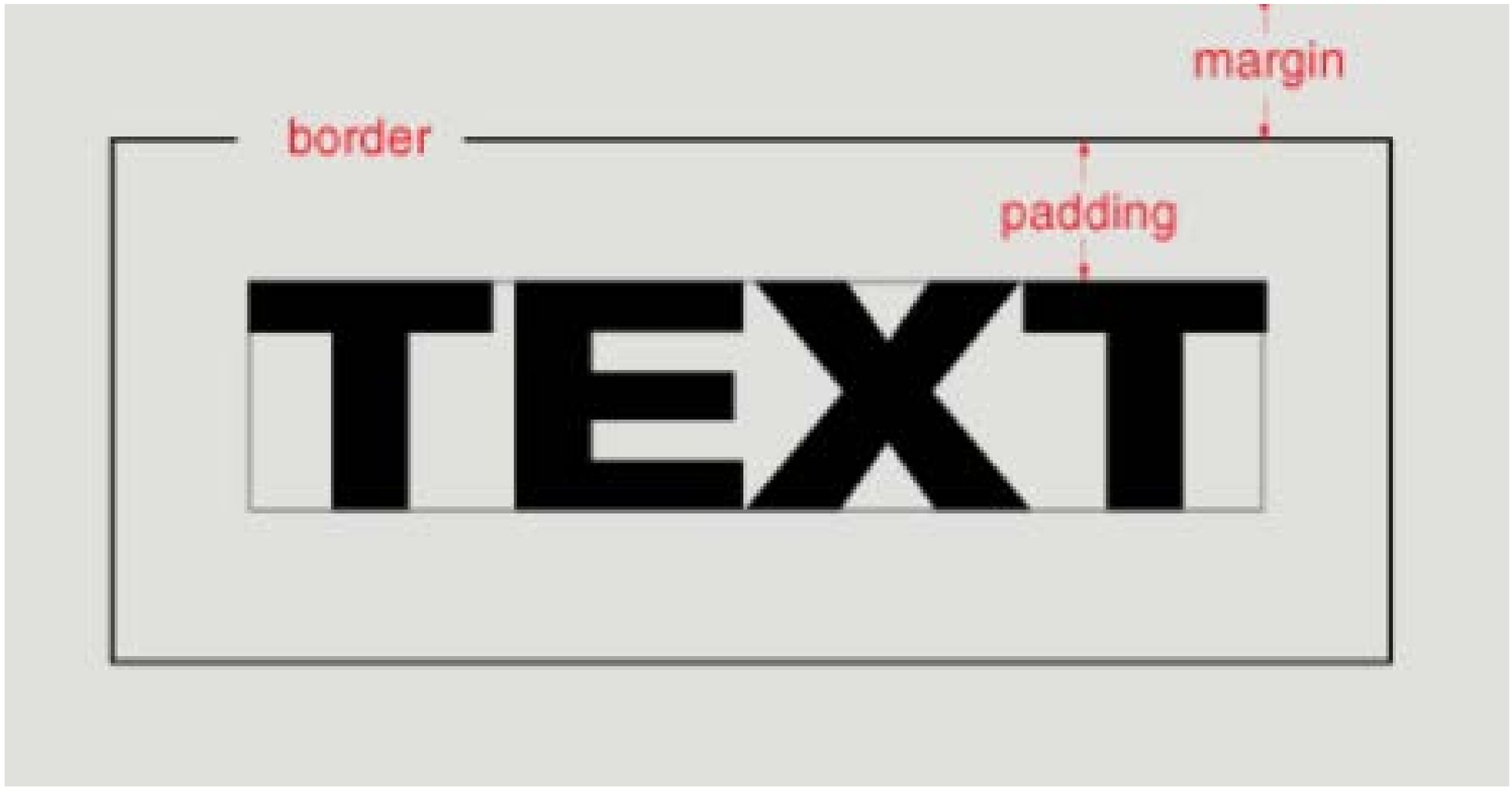
**CONTENT:** stuff in the box

**PADDING:** bubble wrap and packing peanuts

**BORDER:** sides of the box

**MARGIN:** space between multiple boxes

# CSS BOX MODEL



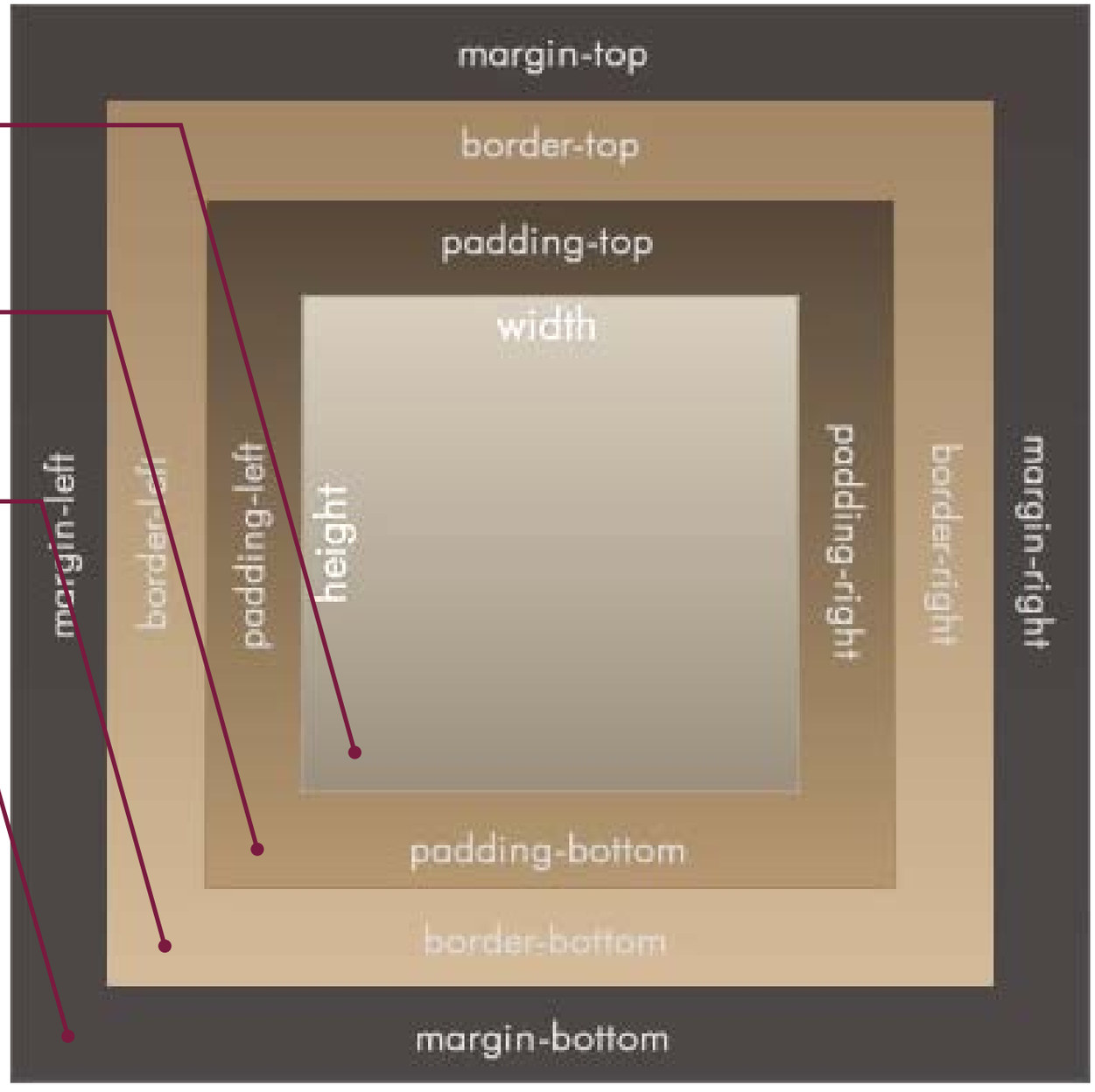
# CSS BOX MODEL

Content

Padding

Border

Margin

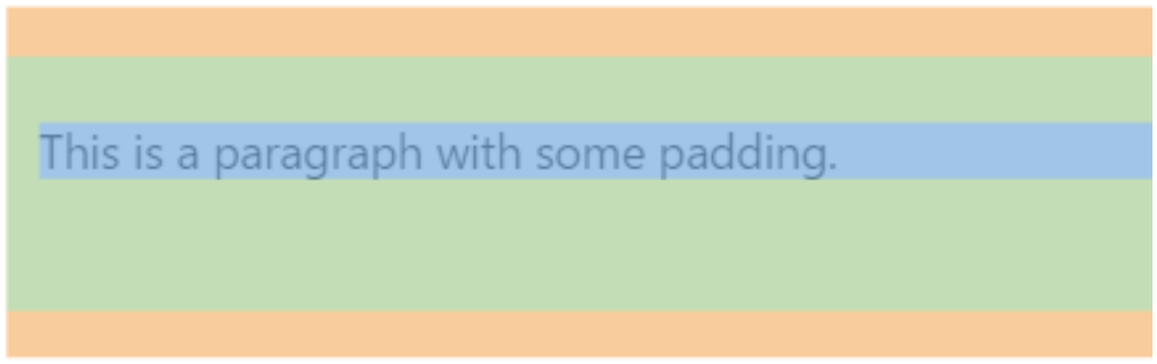


# PADDING

Padding creates space **inside** an element.

Padding affects how far content is from the border.

```
p {
    padding-top: 20px;
    padding-right: 5px;
    padding-bottom: 40px;
    padding-left: 10px;
}
```




Shorter way:

```
p {
    padding: 20px 5px 40px 10px;
}
```

# PADDING

Padding is useful for moving content away from the edges of its container.

This is a paragraph with zero padding (default)

This is a paragraph with 1em padding

# PADDING

If **top/bottom** and **left/right** padding match...

```
p {
    padding-top: 20px;
    padding-right: 10px;
    padding-bottom: 20px;
    padding-left: 10px;
}
```

Combine them!

```
p { padding: 20px 10px; }
```

# PADDING

If **all** padding matches...

```
p {
    padding-top: 20px;
    padding-right: 20px;
    padding-bottom: 20px;
    padding-left: 20px;
}
```

Combine EVEN MORE!

```
p { padding: 20px; }
```

# PADDING

Padding can be applied only to the top, only to the bottom, and so on – or any combination of those:

```
p {
    padding-left: 40px;
}

p {
    padding-top: 20px;
    padding-right: 10px;
}
```

# MARGIN

Margin creates space **outside** an element.

- Same abbreviation style and rules as padding

```
p {
    margin-top: 20px;
    margin-right: 5px;
    margin-bottom: 40px;
    margin-left: 10px;
}
```

Is the same as

```
p { margin: 20px 5px 40px 10px; }
```

# MARGIN

You can give `margin` a **negative** value to shift elements in the opposite direction.

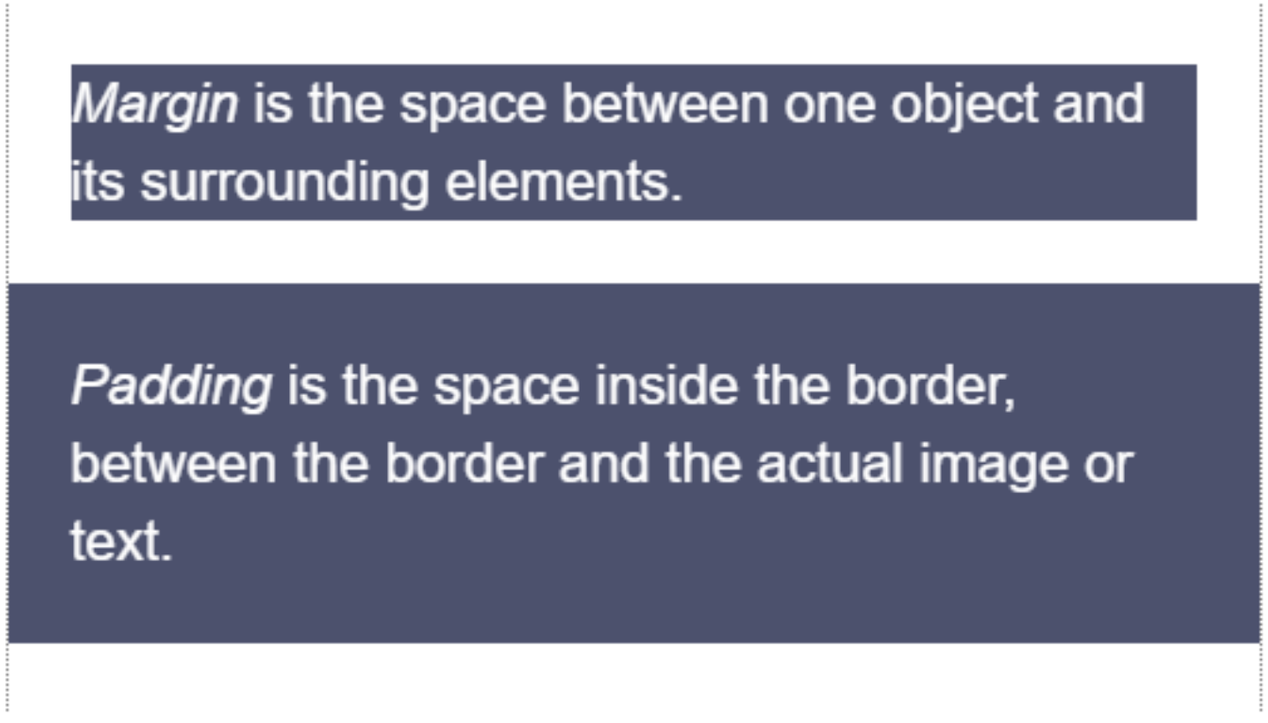
```
p {
    margin-top: -20px;
}
```

This may result in overlapping text!

# MARGIN VS. PADDING

Use `margin` to separate the element from the things that are around it.

Use `padding` to move the element away from the edges of the block.



# BORDER STYLES

Between margin and padding, you can set a `border`

Values are separated with spaces, in this order:

- Width (usually in pixels, but can be em)
- Border style (solid, dotted, dashed, etc)
- Color

```
p {
    border: 2px dotted #ff0000;
}
```

# BORDER STYLES

## Border styles:

| | |
|---|---|
| `solid` | Solid line. |
| `dotted` | Series of dots. |
| `dashed` | Series of dashes. |
| `double` | Two solid lines. |
| `groove` | Representation of a carved groove. Opposite of `ridge`. |
| `ridge` | Representation of an embossed ridge. Opposite of `groove`. |
| `inset` | Representation of an inset depression. Opposite of `outset`. |
| `outset` | Representation of an outset extrusion. Opposite of `inset`. |

# BORDER STYLES

You can set a border on only one side of an element:

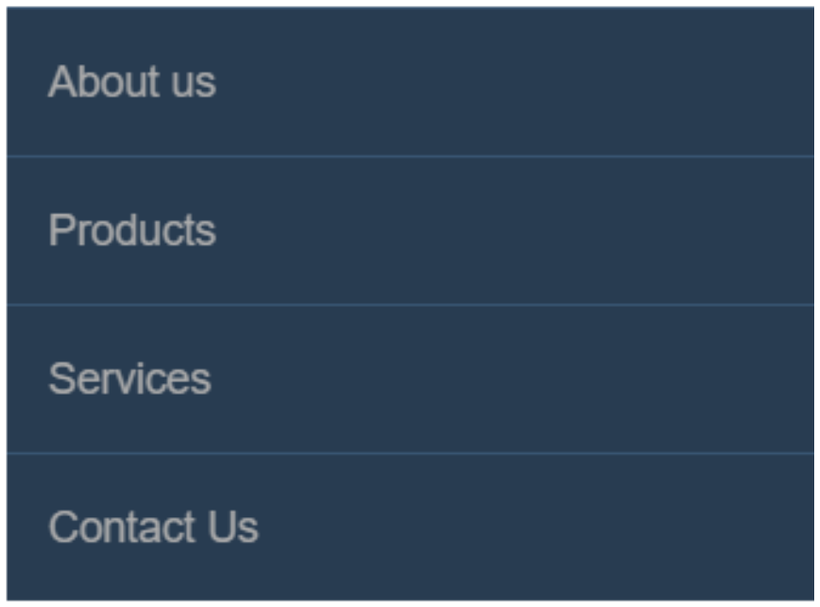
```
h1 { border-bottom: 3px solid black; }
```

## HEADER WITH BORDER BOTTOM

# BORDER STYLES

A common use of `border` is to visually separate list items in a navigation menu.

```
ul {
      list-style : none;
}

li {
      padding: 1em;
      background-color: #283c51;
      border-top: 1px solid #395673;
      color: #adadad;
}
```



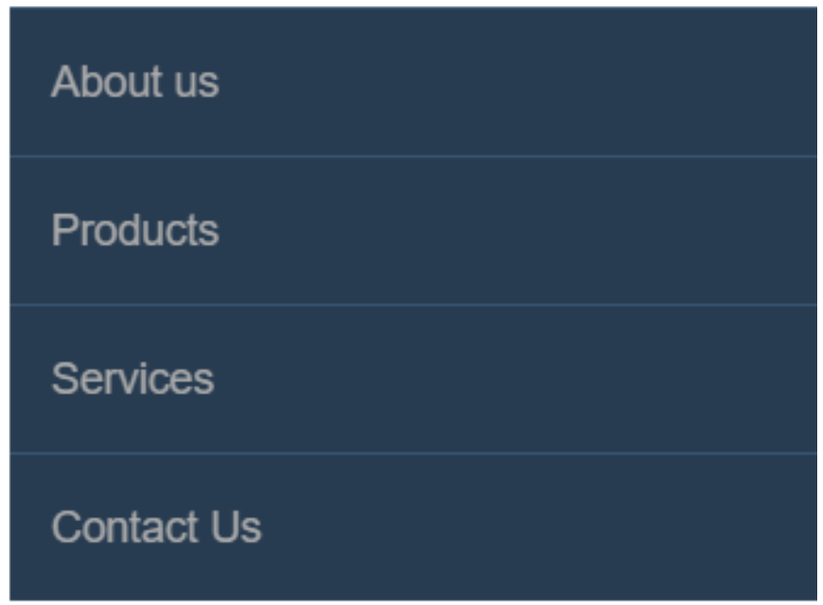

```
<ul>
      <li>About us</li>
      <li>Products</li>
      <li>Services</li>
      <li>Contact Us</li>
</ul>
```

# LIST STYLE

Note that we set

```
ul {
     list-style: none;
}
```

to remove the bullets that appear by default on an unordered list

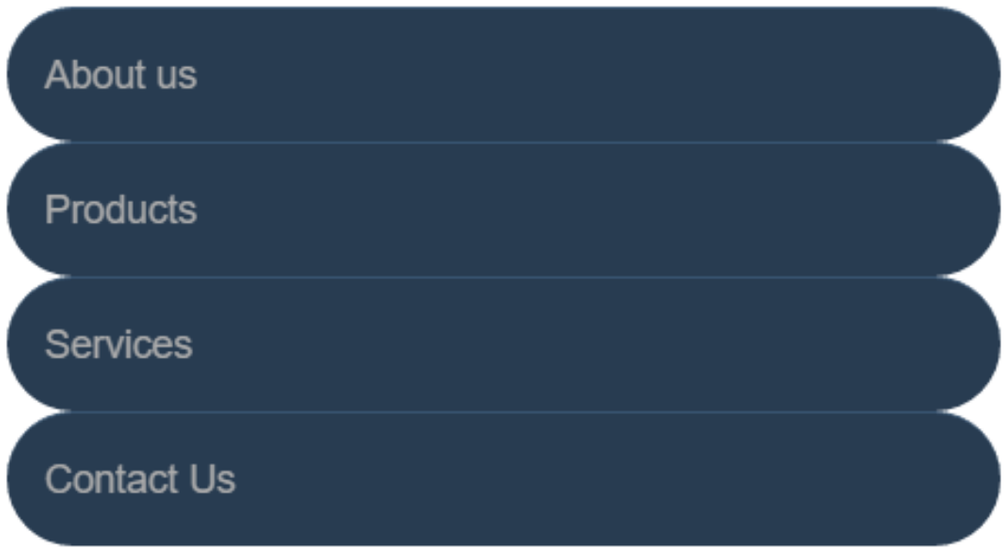


# BORDER RADIUS

To make an element appear curved, use the property `border-radius`

- The value is a number (in px or em) or percentage
- You can use `border-radius` even if you don't explicitly set a `border`

```
li {
    /* same styles... */
    border-radius: 2em;
}
```



# BORDER RADIUS

`border-radius` can be used to create a circle.

- Set `border-radius` to `50%`
- Set `height` and `width` to the same value

```
li {
      border-radius: 50%;
      background-color: black;
      color: white;
      text-align: center;
      height: 3em;
      width: 3em;
      line-height: 3em;
      margin: 5px;
}
```

# BORDER RADIUS

This technique can be used on images to crop them into a circle

- If the image itself doesn't have a square ratio, it will look distorted

```
img {
      border-radius: 50%;
      height: 200px;
      width: 200px;
}
```

# PRACTICE TIME!

# ASSIGNMENT

If you haven't already, create a navigation section for your website.

- Add a **list** of links in your navigation menu

- Make the navigation menu pretty by using padding, margin, border, background color, and other tricks we've learned.
  - ONLY style lists that are in the nav menu – not lists on the rest of the page
  - Bonus points: style the last or first item in the nav menu differently using **pseudo-classes**

- Give your page "breathing room" with padding and/or margin.

# BLOCK VS. INLINE ELEMENTS

# BLOCK ELEMENTS

- Expand naturally to fill their parent container
  - Takes up a “full line”
- Can have margin and/or padding
- Can have height and/or width
- By default, will be placed **below** previous elements in the markup

# <> BLOCK ELEMENTS

# <> BLOCK ELEMENTS

Examples of block elements:

- Headings `<h1>`...`<h6>`
- Paragraphs `<p>`
- Lists `<ul>`, `<ol>`

# INLINE ELEMENTS

- Flow along with text content
- Only take up as much space as necessary
- Ignore width and height properties
- Margin and padding only pushes other elements away horizontally, not vertically
- Top and bottom margin/padding is ignored

# <> INLINE ELEMENTS

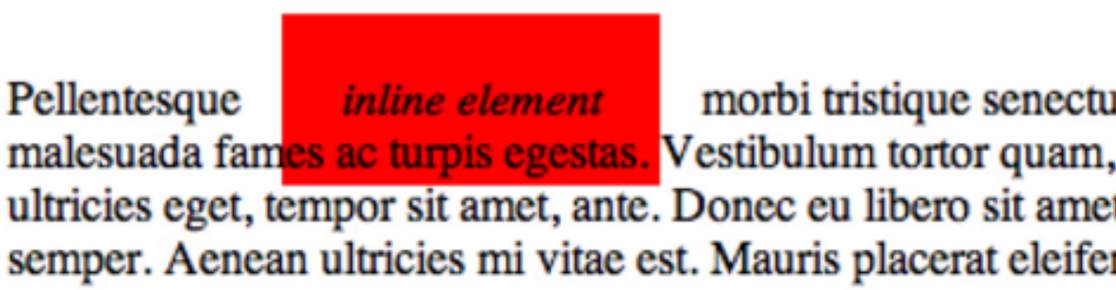
## INLINE ELEMENTS FLOW WITH TEXT

PELLENTESQUE HABITANT MORBI TRISTIQUE SENECTUS ET NETUS ET MALESUADA FAMES AC TURPIS EGESTAS. VESTIBULUM INLINE ELEMENT VITAE, ULTRICIES EGET, TEMPOR SIT AMET, ANTE. DONEC EU LIBERO SIT AMET QUAM EGESTAS SEMPER. AENEAN ULTRICIES MI VITAE EST. MAURIS PLACERAT ELEIFEND LEO.

# <> BLOCK & INLINE ELEMENTS

Examples of inline elements:

- Links `<a>`
- Font emphasis `<em>`
- Font bold `<strong>`

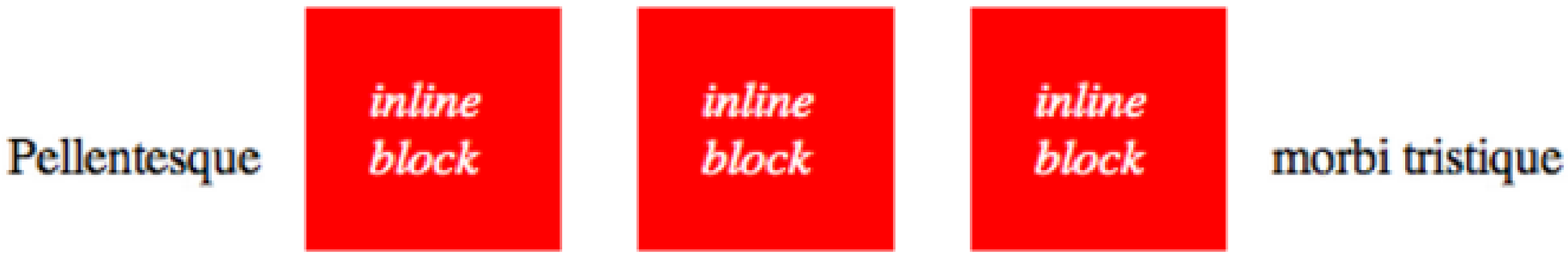


# <> INLINE BLOCK

## INLINE-BLOCK ELEMENT

- Is a hybrid of inline and block
- Takes up width and height like block-level elements
- Flows with content
- Can have margin and padding
- Examples of inline-block elements:
  - Image `<img />`

# <> INLINE BLOCK




senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

# <> DISPLAY

You can change whether or not any element is block, inline, or inline-block by using the CSS `display` property.

- This means we can do some neat things!

```
li {
    display: inline-block;
}
```

# PRACTICE TIME!

# ASSIGNMENT

Update your navigation to a horizontal menu using CSS.

- Give the `li` elements a `display` property of either `inline` or `inline-block`. Which looks better? Why do you think that is?
- Update your styles so that they look nice in the new orientation

Create a link that looks like a button.

- Style the button differently on hover and click
- What happens if you want to put two “button-style” links next to one another?
- What if you want the two buttons to be the same width even when they have different text?

<html>

# (MORE) HTML ELEMENTS

# <SPAN> ELEMENTS

```
<span></span>
```

A `<span>` is a **generic inline element**

- No default style

- Used to style inline content

# <DIV> ELEMENTS

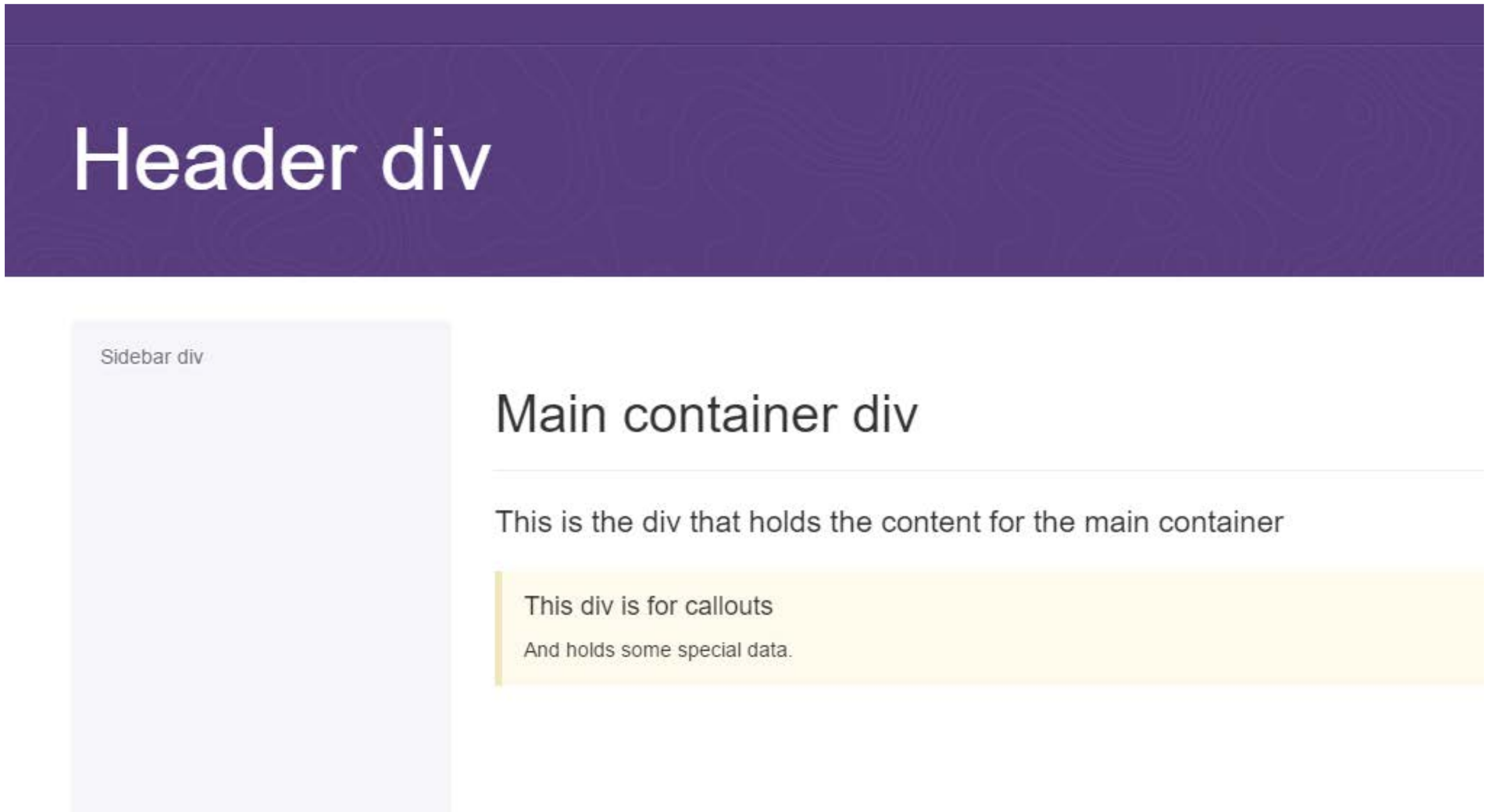
```
<div></div>
```

A `<div>` is a **generic block element**

- No default style

- Heavily used as a wrapper for other elements, to create complex layouts

# <DIV> LAYOUT EXAMPLE



# <DIV> LAYOUT EXAMPLE

```
<div class="header">
    <h1>Header div</h1>
</div>

<div class="row">

    <div class="sidebar">
        Sidebar div
    </div>

    <div class="main">

        <h2>Main container div</h2>
        <p>This is the div that holds the content for the main container</p>

        <div class="callout">
            <h4>This div is for callouts</h4>
            <p>And holds some special data.</p>
        </div>
    </div>
</div>
```

# WHY USE DIV OR SPAN?

Both `div` and `span` really need something extra to be useful, since they have no presentation style by default.

- Bonus: you don't need to "reset" them before making them fit your design (like `ul` or `p`)

- But... how do you style them anyway?

# ID & CLASS SELECTORS

# CLASSES AND IDS

CSS lets us target **all** paragraphs like this:

```
p {
    font-size: 20px;
}
```

But what if we want to style only **some** paragraphs?

# CLASSES AND IDS

You can add `class` and `id` to any HTML element to identify it for styling.

- You decide the `class` and `id` values – be descriptive!

```
<p class="important">Big text</p>

<p class="anyLettersOrNumbersOr_Or-">Still
totally valid</p>
```

# CLASSES AND IDS

Adding a `class` or `id` does nothing to an element by default.

- Classes and ids don't have any styling information by themselves
- They require you to add CSS selectors if you want styling to be applied

# CLASSES AND IDS

**Multiple** elements can have the same `class`

- A class is like a barcode – all of the same products have the same barcode


Only **one** element per page can use the same `id`

- An id is like a serial number – it uniquely identifies one specific instance of a product

# CLASS SELECTORS IN CSS

- In CSS, target a `class` with a **`period`**

- Will style **all** types of elements that have that `class`:

```
.kittens { color: gray; }

<p class="kittens">This will be gray.</p>

<div class="kittens">This will be gray too.</div>
```

# CLASS ATTRIBUTES

Elements can have **multiple** classes, separated by a space:

```
<p class="important margin-sm">Big text with a
margin</p>


.important { font-size: 20px; }
.margin-sm { margin: 5px; }
```

# CLASS SELECTORS IN CSS

Child selectors work with classes:

```
.card p { padding: 16px; }
```

“Any paragraph that is **inside** an element with a `class` of `card` gets 16px of padding.”

```
<div class="card">
  <h2>This will not get padding</h2>
  <p>But this will</p>
</div>
```

# ID ATTRIBUTES

- An `id` can only be used **once** per page
- Elements **cannot** have multiple `id` attributes

```
<div id="mainContent">
     <!-- This better be the only main -->
</div>
```

# ID SELECTORS IN CSS

In CSS, target an id with a **hash**:

```
#kittenContainer {
    color: gray;
}
```

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

# IDs for anchoring

If you put a hash followed by the element's `id` in the URL, the browser will **jump** to that location on the same page:

```
<a href="#kittenContainer">Proceed directly to kittens</a>
```

# ID ATTRIBUTES

**Q:** What horrible thing will happen if you use an `id` twice on the same page?

**A:** Well...actually nothing.

- But your page won't validate
- Jump links will go to whatever `id` appears first
- And any Javascript that needs to locate that specific element will fail

# MIXING CLASS AND ID ATTRIBUTES

An element can have both `id` and `class` attributes.

```
<div id="puppyContainer" class="small fluffy"></div>

<div id="birdContainer" class="small feathery"></div>
```

# HOW TO CHOOSE - CLASS OR ID?

If you think it's likely or possible that you'll want to apply the same style to multiple things, definitely use `class`

If your element is guaranteed to be the only one on the page, you can use `id` – or you can still use `class`

If your element needs to be linked to directly, use `id`

# CLASS COMPONENTS

The most common use of classes is to define reusable components.


Located two hours south of Sydney in the Southern Highland of New South Wales...

SHARE LEARN MORE


Largest monolith (geological feature consisting of a single massive stone or rock) in the world.

SHARE LEARN MORE

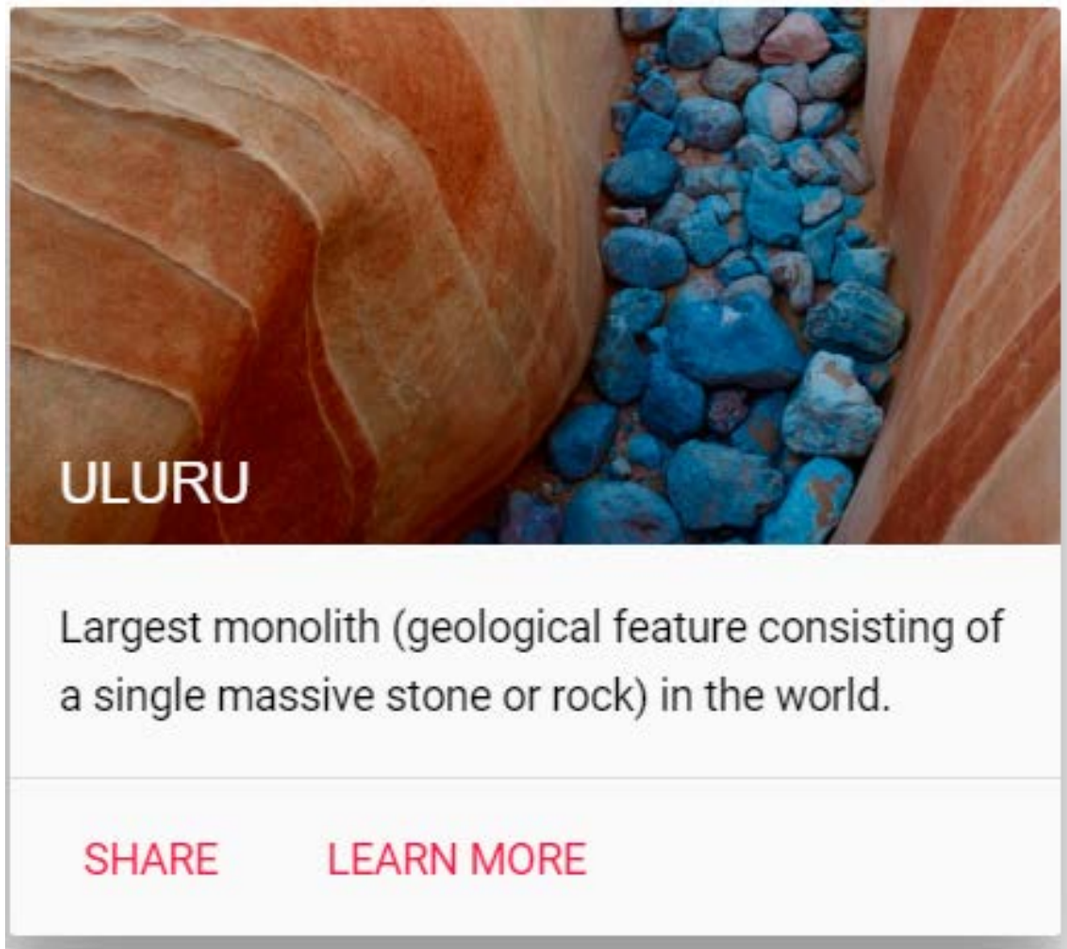
# CLASS COMPONENTS

A component is an outline defined in HTML that will have the same markup every time it's used, but with different content inside it.

```
<div class="card">

    <div class="image">
        <h2></h2>
    </div>


    <p></p>

    <div class="action-bar">
        <a></a>
        <a></a>
     </div>

</div>
```

# CHILDREN IN CLASS

```
.card {
        display: inline-block;
        width: 344px;
        height: 304px;
}
.card p {
        padding: 16px;
        margin: 0;
        line-height: 1.6;
}
.card .action-bar {
        padding: 0 8px;
        border-top: 1px solid #E0E0E0;
        height: 52px;
}
```




See a live demo

# PRACTICE TIME!

# ASSIGNMENT

Give an element on your page a descriptive `class`

- Apply a special style using a CSS `class` selector
- Style a child element of this element

Create another class and apply it to **two different** types of elements

- Bonus points: apply to an element that already has a class. What happens if the styles conflict? How would you make sure the result is what you want?

Assign an `id` to an element on your page

- Apply a unique style using an `id` selector
- Create a link in your nav that jumps to that element

# “HOMEWORK”

- Practice!

- Optional: read chapter 8 of *HTML and CSS: Design and Build Websites*


- Try playing with this interactive demo of the CSS box model