CSS 2

Programming with Web Technologies

Previously, in CSS 1

Learned about some CSS properties, and different ways to include CSS rules in our documents

Looked in detail at fonts, including using web fonts

Learned about different selectors

Looked at the CSS box model

Touched on CSS units

Today, in CSS 2

Looking at some more selectors

Positioning elements on the page

New CSS features - Animations, Filters, Transformations, Transitions

Responsive Design using Media Queries

More form attributes

Pseudo Class Selectors

In CSS1 we looked at a number of selectors, but all of them had one thing in common - they all related to information that was given in the HTML structure, such as the element type, id, class and attributes

But what about characteristics that aren't represented directly in the markup?

- A link that hasn't been visited
- A link that has been visited
- The mouse cursor is hovering over an element
- Whether an input has focus

Pseudo class selectors can be used to select elements in these special states

Pseudo Class Selectors

There are a large number of pseudo classes available, far too many to cover here, but a complete list can be found on MDN

Pseudo class selectors take the form

```
selector:pseudo-class { property: value; }
```

Where `selector` is any valid CSS selector, and `pseudo-class` is a pseudo class appropriate for the element selected

Pseudo Class Selectors

```
a:visited { }
```

Matches an anchor element that has been visited

```
div.important:hover { }
```

Matches a `div` element with the class important, that is being hovered over

```
input[type="text"]:focus { }
```

Matches a text `input` field that the client has clicked into

```
p:nth-child(5) { }
```

Matches all `p` elements that are the 5th child of their parent

CSS Positioning

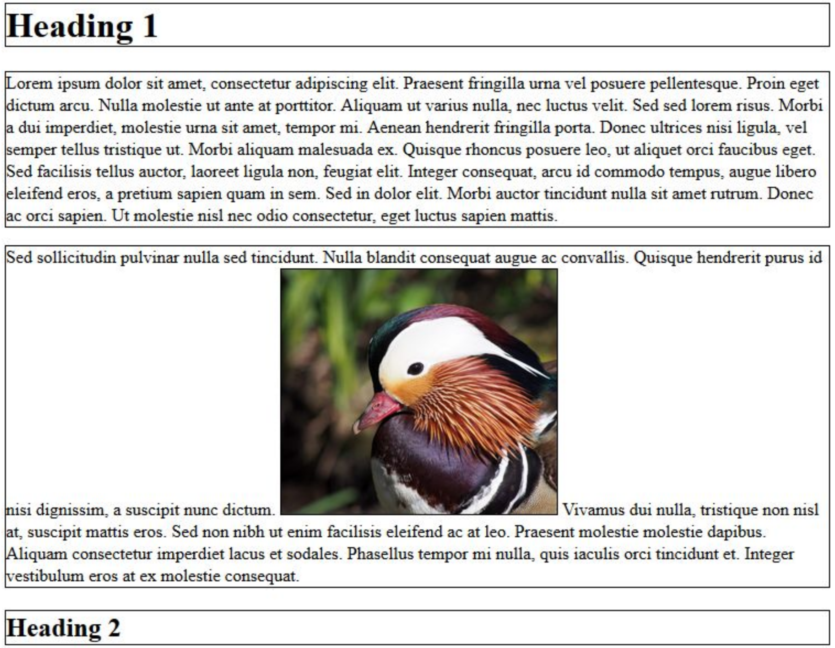
Up until now, all of our CSS has been positioned in **normal flow**

What this means for **block** elements is that they have been laid out vertically as boxes placed one after the other, top-to-bottom. Distance between other box is based on the margin properties of each, and the left edge of each box touches the left edge of its parent

For our **inline** elements, they have been laid out horizontally, left-to-right, separated by their margins, padding and borders. Vertical alignment can be by the element tops, bottoms or text baseline

CSS Positioning - Normal Flow

```
<h1>Heading 1</h1>
  <p>
     ...
  </p>
  <p>
    ...
    <img src="duck.png">
    ...
  </p>
<h2>Heading 2</h2>
```



Heading 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent fringilla urna vel posuere pellentesque. Proin eget dictum arcu. Nulla molestie ut ante at porttitor. Aliquam ut varius nulla, nec luctus velit. Sed sed lorem risus. Morbi a dui imperdiet, molestie urna sit amet, tempor mi. Aenean hendrerit fringilla porta. Donec ultrices nisi ligula, vel semper tellus tristique ut. Morbi aliquam malesuada ex. Quisque rhoncus posuere leo, ut aliquet orci faucibus eget. Sed facilisis tellus auctor, laoreet ligula non, feugiat elit. Integer consequat, arcu id commodo tempus, augue libero eleifend eros, a pretium sapien quam in sem. Sed in dolor elit. Morbi auctor tincidunt nulla sit amet rutrum. Donec ac orci sapien. Ut molestie nisl nec odio consectetur, eget luctus sapien mattis.

Sed sollicitudin pulvinar nulla sed tincidunt. Nulla blandit consequat augue ac convallis. Quisque hendrerit purus id nisi dignissim, a suscipit nunc dictum. Vivamus dui nulla, tristique non nisl at, suscipit mattis eros. Sed non nibh ut enim facilisis eleifend ac at leo. Praesent molestie molestie dapibus. Aliquam consectetur imperdiet lacus et sodales. Phasellus tempor mi nulla, quis iaculis orci tincidunt et. Integer vestibulum eros at ex molestie consequat.

Heading 2

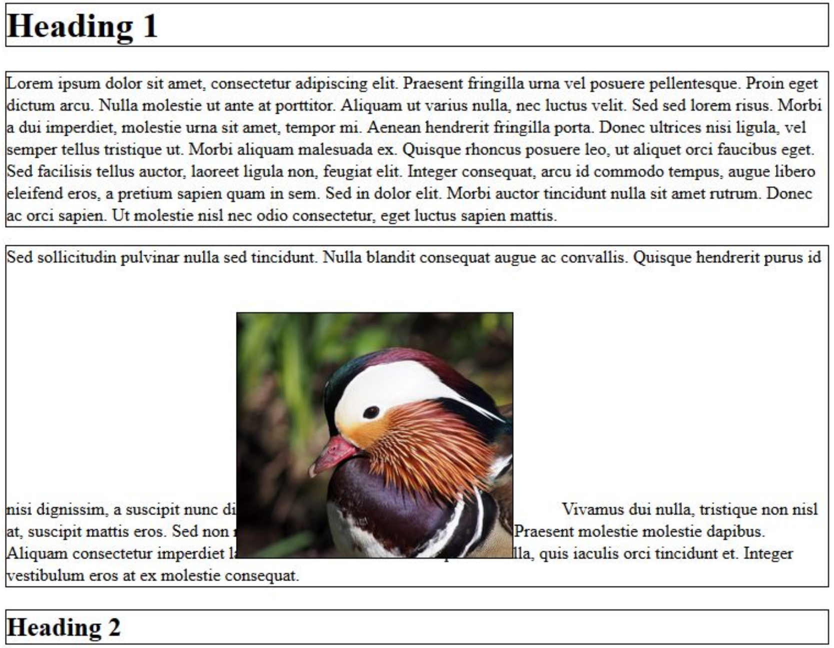
CSS Positioning

The way an element is positioned is controlled by the `position` property, and this can be set to one of 5 values

`static`	(default) Positioned using normal flow
`relative`	Positioned relative to where it would be using normal flow by providing an offset
`absolute`	Removed from normal flow and positioned by coordinates relative to the containing block
`fixed`	A constant position relative (normally) to the window i.e. not affected by scrolling
`sticky`	Toggles between relative and fixed depending on scroll position

CSS Positioning - Relative

```css
img {
  position: relative;
  left: -40px;
  top: 40px;
  /* Could use right or bottom */
}
```



Heading 1

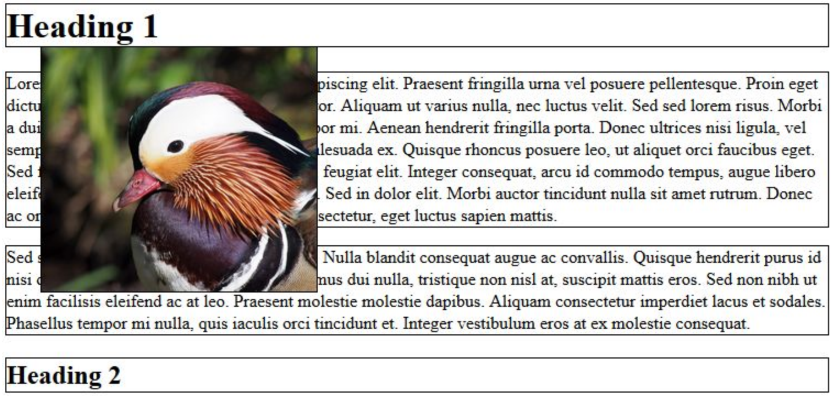
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent fringilla urna vel posuere pellentesque. Proin eget dictum arcu. Nulla molestie ut ante at porttitor. Aliquam ut varius nulla, nec luctus velit. Sed sed lorem risus. Morbi a dui imperdiet, molestie urna sit amet, tempor mi. Aenean hendrerit fringilla porta. Donec ultrices nisi ligula, vel semper tellus tristique ut. Morbi aliquam malesuada ex. Quisque rhoncus posuere leo, ut aliquet orci faucibus eget. Sed facilisis tellus auctor, laoreet ligula non, feugiat elit. Integer consequat, arcu id commodo tempus, augue libero eleifend eros, a pretium sapien quam in sem. Sed in dolor elit. Morbi auctor tincidunt nulla sit amet rutrum. Donec ac orci sapien. Ut molestie nisl nec odio consectetur, eget luctus sapien mattis.

Sed sollicitudin pulvinar nulla sed tincidunt. Nulla blandit consequat augue ac convallis. Quisque hendrerit purus id nisi dignissim, a suscipit nunc di Vivamus dui nulla, tristique non nisl at, suscipit mattis eros. Sed non Praesent molestie molestie dapibus. Aliquam consectetur imperdiet l lla, quis iaculis orci tincidunt et. Integer vestibulum eros at ex molestie consequat.

Heading 2

CSS Positioning - Absolute

```css
img {
  position: absolute;
  left: 40px;
  top: 60px;
  /* Could use right or bottom */
}
```



Heading 1

Lore piscing elit. Praesent fringilla urna vel posuere pellentesque. Proin eget
dictu or. Aliquam ut varius nulla, nec luctus velit. Sed sed lorem risus. Morbi
a dui or mi. Aenean hendrerit fringilla porta. Donec ultrices nisi ligula, vel
semp lesuada ex. Quisque rhoncus posuere leo, ut aliquet orci faucibus eget.
Sed feugiat elit. Integer consequat, arcu id commodo tempus, augue libero
eleif . Sed in dolor elit. Morbi auctor tincidunt nulla sit amet rutrum. Donec
ac or sectetur, eget luctus sapien mattis.

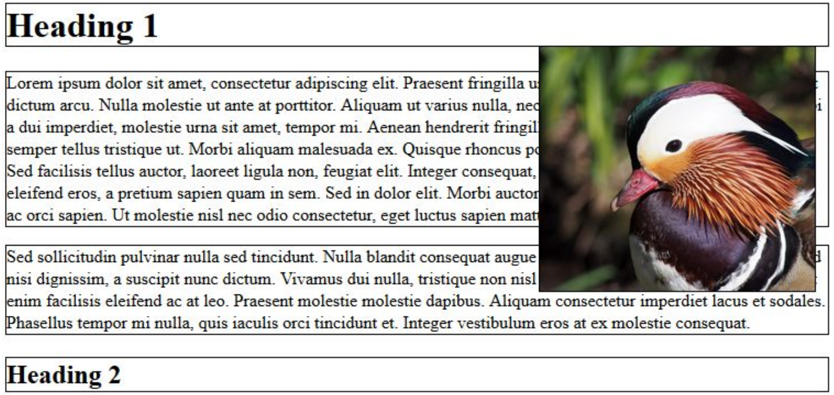
Sed Nulla blandit consequat augue ac convallis. Quisque hendrerit purus id
nisi mus dui nulla, tristique non nisl at, suscipit mattis eros. Sed non nibh ut
enim facilisis eleifend ac at leo. Praesent molestie molestie dapibus. Aliquam consectetur imperdiet lacus et sodales.
Phasellus tempor mi nulla, quis iaculis orci tincidunt et. Integer vestibulum eros at ex molestie consequat.

Heading 2

Absolute positioning positions within the first containing element that is not in **normal flow**. In this case, the paragraph where the image is placed is in the normal flow. The closest container that would make sense would be the body element

CSS Positioning - Fixed

```css
img {
  position: fixed;
  right: 20px;
  top: 60px;
}
```



Heading 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent fringilla u
dictum arcu. Nulla molestie ut ante at porttitor. Aliquam ut varius nulla, nec
a dui imperdiet, molestie urna sit amet, tempor mi. Aenean hendrerit fringil
semper tellus tristique ut. Morbi aliquam malesuada ex. Quisque rhoncus po
Sed facilisis tellus auctor, laoreet ligula non, feugiat elit. Integer consequat,
eleifend eros, a pretium sapien quam in sem. Sed in dolor elit. Morbi auctor
ac orci sapien. Ut molestie nisl nec odio consectetur, eget luctus sapien mat

Sed sollicitudin pulvinar nulla sed tincidunt. Nulla blandit consequat augue
nisi dignissim, a suscipit nunc dictum. Vivamus dui nulla, tristique non nisl
enim facilisis eleifend ac at leo. Praesent molestie molestie dapibus. Aliquam consectetur imperdiet lacus et sodales.
Phasellus tempor mi nulla, quis iaculis orci tincidunt et. Integer vestibulum eros at ex molestie consequat.

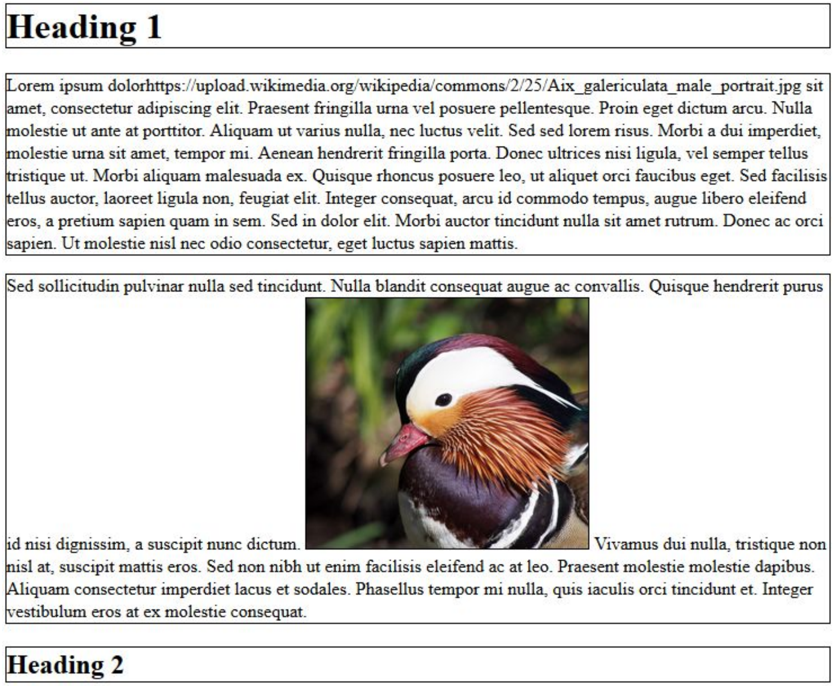
Heading 2

Fixed positioning is used when you want something to always stay at the same position on the screen. A fixed position element will not scroll with the page

CSS Positioning - Sticky

```css
img {
  position: -webkit-sticky;
  position: sticky;
  top: 0px;
}
```

Sticky is difficult to visualize with a static image, but when the image reaches the top of the page due to scrolling, it will 'stick' to it, acting like a fixed positioned element



Heading 1

Lorem ipsum dolorhttps://upload.wikimedia.org/wikipedia/commons/2/25/Aix_galericulata_male_portrait.jpg sit amet, consectetur adipiscing elit. Praesent fringilla urna vel posuere pellentesque. Proin eget dictum arcu. Nulla molestie ut ante at porttitor. Aliquam ut varius nulla, nec luctus velit. Sed sed lorem risus. Morbi a dui imperdiet, molestie urna sit amet, tempor mi. Aenean hendrerit fringilla porta. Donec ultrices nisi ligula, vel semper tellus tristique ut. Morbi aliquam malesuada ex. Quisque rhoncus posuere leo, ut aliquet orci faucibus eget. Sed facilisis tellus auctor, laoreet ligula non, feugiat elit. Integer consequat, arcu id commodo tempus, augue libero eleifend eros, a pretium sapien quam in sem. Sed in dolor elit. Morbi auctor tincidunt nulla sit amet rutrum. Donec ac orci sapien. Ut molestie nisl nec odio consectetur, eget luctus sapien mattis.

Sed sollicitudin pulvinar nulla sed tincidunt. Nulla blandit consequat augue ac convallis. Quisque hendrerit purus id nisi dignissim, a suscipit nunc dictum. Vivamus dui nulla, tristique non nisl at, suscipit mattis eros. Sed non nibh ut enim facilisis eleifend ac at leo. Praesent molestie molestie dapibus. Aliquam consectetur imperdiet lacus et sodales. Phasellus tempor mi nulla, quis iaculis orci tincidunt et. Integer vestibulum eros at ex molestie consequat.

Heading 2

Vendor Prefixes

In the previous slide, you might have noticed the `-webkit-sticky` value was used to set the position property. The `-webkit-` portion of this value is known as a vendor prefix

Like HTML5, CSS3 is a living standard and new features are constantly being introduced. While new features are being developed, browser vendors will release versions of the new feature that developers can try out,

Because the draft standards for features might change during development, the w3c doesn't want vendors to use the official names of features before they are released, so vendors add a prefix to the property name

Vendor Prefixes

The prefixes used by each vendor are standardized. `-moz-` is used by Firefox; `-webkit-` is used by Safari, Chrome and Opera; `-o-` was used by older versions of Opera; and `-ms-` is used by Internet Explorer and Edge

Not all browser support all CSS features, even with prefixes. Even if they do, the syntax may be slightly different than the proposed standard

Visit the website caniuse.com to search features and see vendor support. Documentation for vendor-prefixed versions can be found on caniuse, or w3schools pages

CSS Positioning - Float

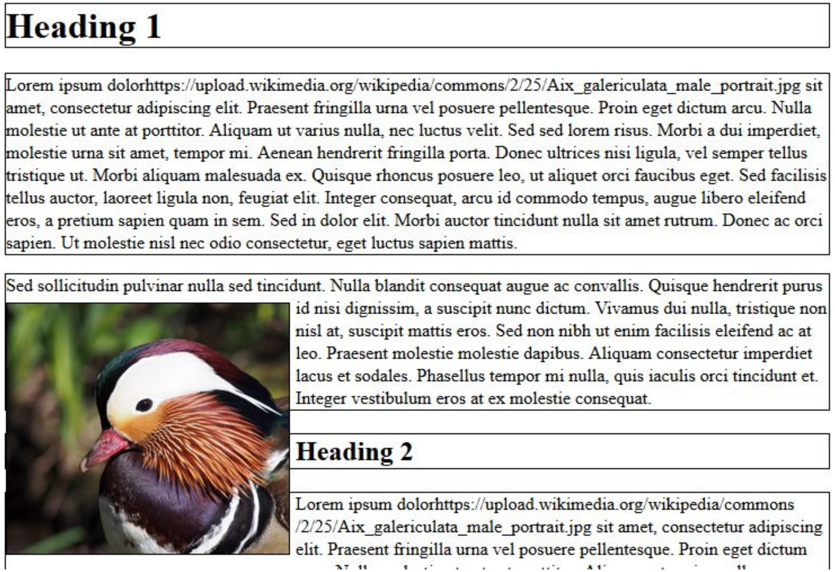
Sometimes, we want to be able to remove elements from the normal flow. This is known as **floating** an element. When an element is floated, it moves to the left or right side of its container, and all other content will flow around it. The `float` property controls the floating state of an element

After an element has been floated, all other content in the normal flow will flow around the element. If this is not what is wanted, you can correct this by clearing the float with the `clear` property

CSS Positioning - Float

```css
img {
  float: left;
  margin: 5px 5px 5px 0px;
}
```

Even though the image has been floated, we can still use box-model properties such as margin to affect how they interact with other elements. In this case a margin has been added to keep the wrapped text a short distance from the image



Heading 1

Lorem ipsum dolorhttps://upload.wikimedia.org/wikipedia/commons/2/25/Aix_galericulata_male_portrait.jpg sit amet, consectetur adipiscing elit. Praesent fringilla urna vel posuere pellentesque. Proin eget dictum arcu. Nulla molestie ut ante at porttitor. Aliquam ut varius nulla, nec luctus velit. Sed sed lorem risus. Morbi a dui imperdiet, molestie urna sit amet, tempor mi. Aenean hendrerit fringilla porta. Donec ultrices nisi ligula, vel semper tellus tristique ut. Morbi aliquam malesuada ex. Quisque rhoncus posuere leo, ut aliquet orci faucibus eget. Sed facilisis tellus auctor, laoreet ligula non, feugiat elit. Integer consequat, arcu id commodo tempus, augue libero eleifend eros, a pretium sapien quam in sem. Sed in dolor elit. Morbi auctor tincidunt nulla sit amet rutrum. Donec ac orci sapien. Ut molestie nisl nec odio consectetur, eget luctus sapien mattis.

Sed sollicitudin pulvinar nulla sed tincidunt. Nulla blandit consequat augue ac convallis. Quisque hendrerit purus id nisi dignissim, a suscipit nunc dictum. Vivamus dui nulla, tristique non nisl at, suscipit mattis eros. Sed non nibh ut enim facilisis eleifend ac at leo. Praesent molestie molestie dapibus. Aliquam consectetur imperdiet lacus et sodales. Phasellus tempor mi nulla, quis iaculis orci tincidunt et. Integer vestibulum eros at ex molestie consequat.

Heading 2

Lorem ipsum dolorhttps://upload.wikimedia.org/wikipedia/commons/2/25/Aix_galericulata_male_portrait.jpg sit amet, consectetur adipiscing elit. Praesent fringilla urna vel posuere pellentesque. Proin eget dictum

CSS Filters

Filters, put simply, are graphical effects that are applied to elements after they have been placed on the page, that change their appearance in a number of ways

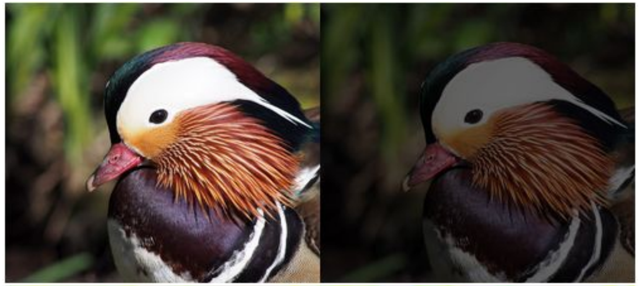
Filters are most commonly used on images due to the supported operations, which include blurring, contrast shifting, brightness adjustment and greyscaling

Controlled with the `filter` property. More than one effect can be specified in the same statement, and will be evaluated left-to-right

CSS Filters

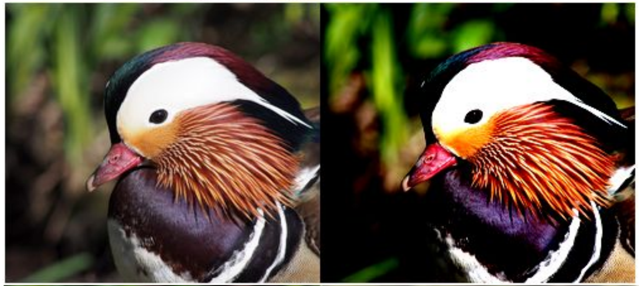
```css
filter: brightness(40%);
```

Decrease the brightness of an image to a percentage of the original



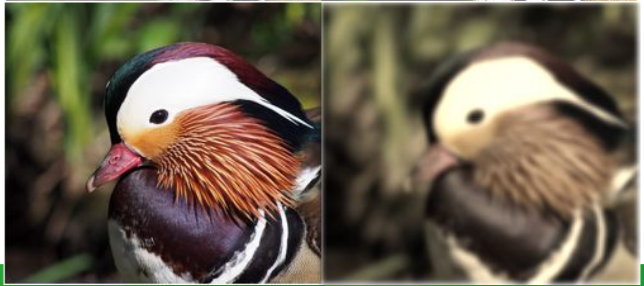
```css
filter: contrast(200%);
```

Tweak the contrast to over-saturate an image



```css
filter: blur(2px) sepia(70%);
```

Slightly blur an image, then apply a strong sepia filter to it



CSS Transitions

Transitions change CSS properties over a time period, which can give a smoother feeling to a user experience. They are simple to specify as you just indicate which property to watch for, and the duration the change should take, then the browser will calculate the intermediate steps for you

Transitions don't run immediately, they run once the property they are watching has changed. You will typically see them used with pseudo class selectors like `:hover`, or with JavaScript

Controlled with the `transition` property. As with filters, more than one property can be watched in the same statement

CSS Transitions

```
#input[type=text] {
    width: 100px;
    transition-property: width;
    transition-duration: 2s;
    transition-timing-function: linear;
    transition-delay: 0.5s;
}

#input[type=text]:hover {
    width: 200px;
}
```

0.5 seconds after a text input gets focus, it will expand to 200px wide over 2 seconds

```
#identifier {
    width: 50px;
    transition: width 2s ease;
}

#identifier:hover {
    width: 200px;
}
```

When hovered over, this element will widen by 150px over 2 seconds, with the change happening slightly faster at the start and end of the transition.

CSS Transformations

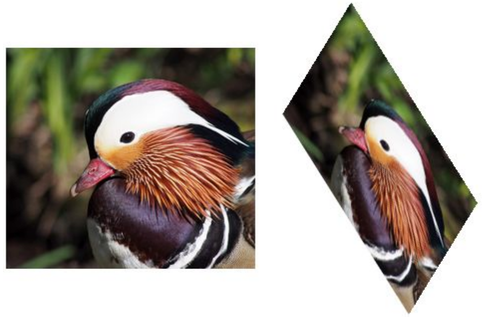
Transformation allow for HTML elements to be visually transformed in 2D or 3D, by rotating, scaling, translating or skewing. 2D operations are very well supported by all browsers, but 3D transforms still have a number of small bugs across platforms

Transforms are run immediately on page load, but can be used in conjunction with a transition or animation to accomplish a smooth or timed behaviour

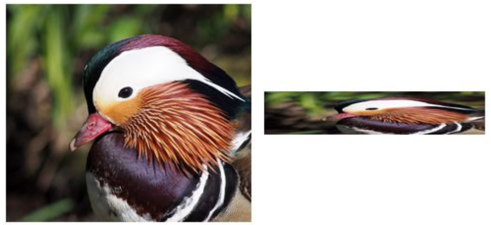
Controlled with the `transform` property. Multiple transformations can be specified at once using the `matrix()` or `matrix3d()` functions

CSS Transformations

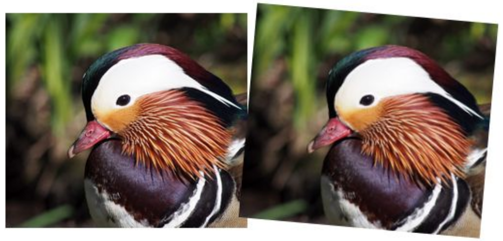
```
transform: rotate3d(1, 1, 1, 75deg);
```
Rotate 75 degrees on all 3 axis



```
transform: scale(0.9, 0.2);
```
Scale the image to 90% width, 20% height



```
transform: rotate(5deg);
```
Rotate the image 5 degrees clockwise



CSS Animations

Up until recently, if we wanted to move something on the page we needed to do it using JavaScript. This could be fairly inefficient, leading to increased power consumption on mobile devices, and a general bad time for the programmer

CSS Animations simplify our lives by removing the JavaScript and efficiency issues and leaving those up to the browser. If CSS animations aren't enough, you can still use JavaScript to augment them

Animations are made up of a `@keyframes` block, as well as a collection of `animation-*` properties to control the behaviour of the defined animation

CSS Animations

```
h1 {
  animation-duration: 3s;
  animation-name: slide-right;
}

@keyframes slide-right {
  from {
    margin-left: 100%;
    width: 300%
  }

  to {
    margin-left: 0%;
    width: 100%;
  }
}
```

This animation starts with its target off screen, accomplished by setting its left margin to 100%, then slides towards the left of the screen until its left side is touching the left of its container.

The h1 element in this case uses this animation and executes it over the course of 3 seconds

CSS Animations

```
@keyframes hrmm {
  from  {
      width: 50px;
      margin-left: 0px;
      background-color: blue;
  } 25% {
      width: 150px;
      margin-left: 0px;
  } 50% {
      width: 50px;
      margin-left: 100px;
      background-color: red;
  } 75% {
      width: 150px;
      margin-left: 100px;
  } to  {
      width: 50px;
      margin-left: 200px;
      background-color: green;
   }
}
```

```
#blue {
  width: 50px;
  height: 50px;
  background-color: blue;
  animation-duration: 3s;
  animation-name: hrmm;
  animation-iteration-count: infinite;
  animation-direction: alternate;
}

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

Any guesses as to what this will do?

Responsive Design

A problem that you will encounter very early on in your web development is that there are a lot of different types of devices with different displays that all want to able to view your site

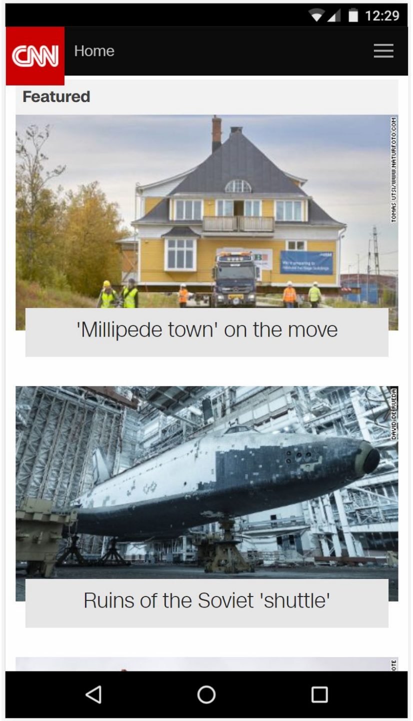
These can range from different sizes of computer screens and projectors, through to mobile phones and tablets, and all the way down to printers and screen readers

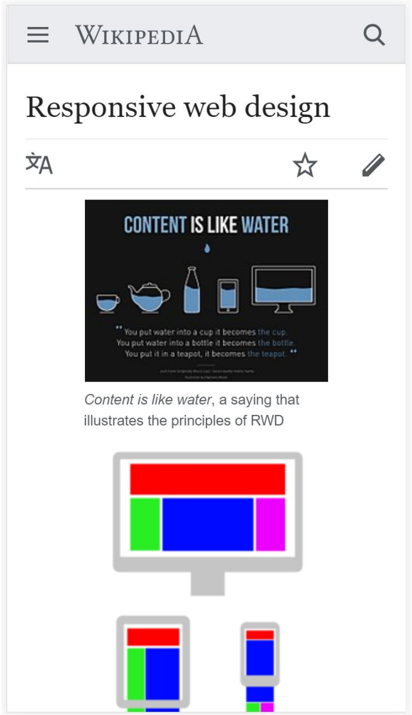
How can we design our pages so that all of these devices can consume our content

Responsive Design

One solution to this problem is to develop a separate webpage for each different class of device. This was especially popular for mobile friendly versions of websites, but has started to fall out of favor

A better solution is to use CSS to alter our page layout depending on what sort of device is viewing the page, and what capabilities it has. This is known as Responsive Design, or any combination of Fluid, Flexible, Adaptive and Elastic layouts




WIKIPEDIA
Responsive web design
CONTENT IS LIKE WATER
"You put water into a cup it becomes the cup.
You put water into a bottle it becomes the bottle.
You put it in a teapot, it becomes the teapot."
Content is like water, a saying that illustrates the principles of RWD

Media Types

CSS2 introduced the ability to conditionally apply CSS rules based on the type of device that was accessing the page. These took the following form:

```
@media type {
  selector {
    property: value;
  }
  …
}
```

Where the `type` was one of the predefined categories of device, such as `all`, `speech`, `print` or `screen`

Media Queries

Media rules were a huge improvement over recreating the site for each type of device, but they suffered from a big problem - not all devices in a category behave the same. Some phones have small screens, others don't; some displays are wide, some are narrow

CSS3 extended the idea of media types with media queries - An extended query language that allows CSS to inspect the **capabilities** of the viewing device rather than just its category

```
@media screen and (min-width: 500px) and (max-width: 699px)
{...}
```

Media Queries

With these more complex queries, **breakpoints** can be defined, which mark sizes of display where the design will switch. This is easily visible on many modern webpages when resizing the window

```
@media print { /* Printer specific styles */ }
@media screen and (min-width: 500px) and (max-width: 699px)
{ /* Small screen */ }
@media screen and (min-width: 700px) and (max-width: 1199px)
{ /* Medium screen */ }
@media screen and (min-width: 1200px) { /* Big screen */ }
```

Media Queries

Media queries can also be used in link statements, allowing CSS file to be loaded only if needed, improving the loading speed of a web page

```
<link rel="stylesheet"
media="screen and (min-width: 641px) and (max-width: 800px)"
href="ipad.css">
```

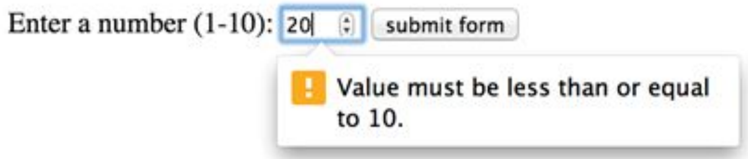
They can also be used inside CSS files to import smaller files, allowing device specific CSS to be kept in its own file, but still be linked to the main CSS file

```
@import url(color.css) screen and (color);
```

HTML Form Validations

In the bad old days of HTML, validating form input had to be done with JavaScript if it was done at all. If you wanted to restrict someone to entering just numbers, you had to check every character as they typed and delete the non-number characters as they appeared

With HTML5, we don't have to do that anymore. You will have noticed that there are now many different types of input that are designed to accept certain things. If you tried some of these, you may have noticed some messages about invalid input

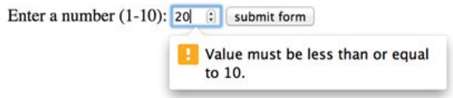

Enter a number (1-10): 20

Value must be less than or equal to 10.

HTML Form Validations

These input types and their auto-validation is great, but what if we want to get more precise and restrict the value, not just the type? Well fortunately we (likely) won't have to resort to JavaScript, though that is always an option

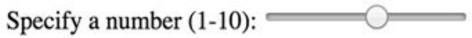
To restrict numeric input, we can use the `min` and `max` attributes



Value must be less than or equal to 10.

Enter a number (1-10):

```
<input type="number" name="quantity" min="1" max="10">
```



Specify a number (1-10):

```
<input type="range" name="rangeNumber" min="1" max="10">
```

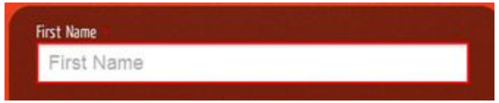
HTML Form Validations

What about enforcing the maximum and minimum length of input? We can use the `size` and `maxlength` attributes

```
<input type="text" size="4" maxlength="4">
```

When we are restricting input options, it is useful to give users a hint as to what sort of values are expected. We can do this using `placeholder` text

```
<input type="text" name="fname" placeholder="First Name">
```



Further Reading

- MDN Pseudo Classes
- MDN CSS Transitions
- MDN CSS Transforms
- 'Just Add Water' CSS Animations
- MDN CSS Animations
- MDN Animatable CSS Properties
- w3schools Responsive Web Design
- MDN Media Queries
- MDN Input Tag & Attributes