The box model

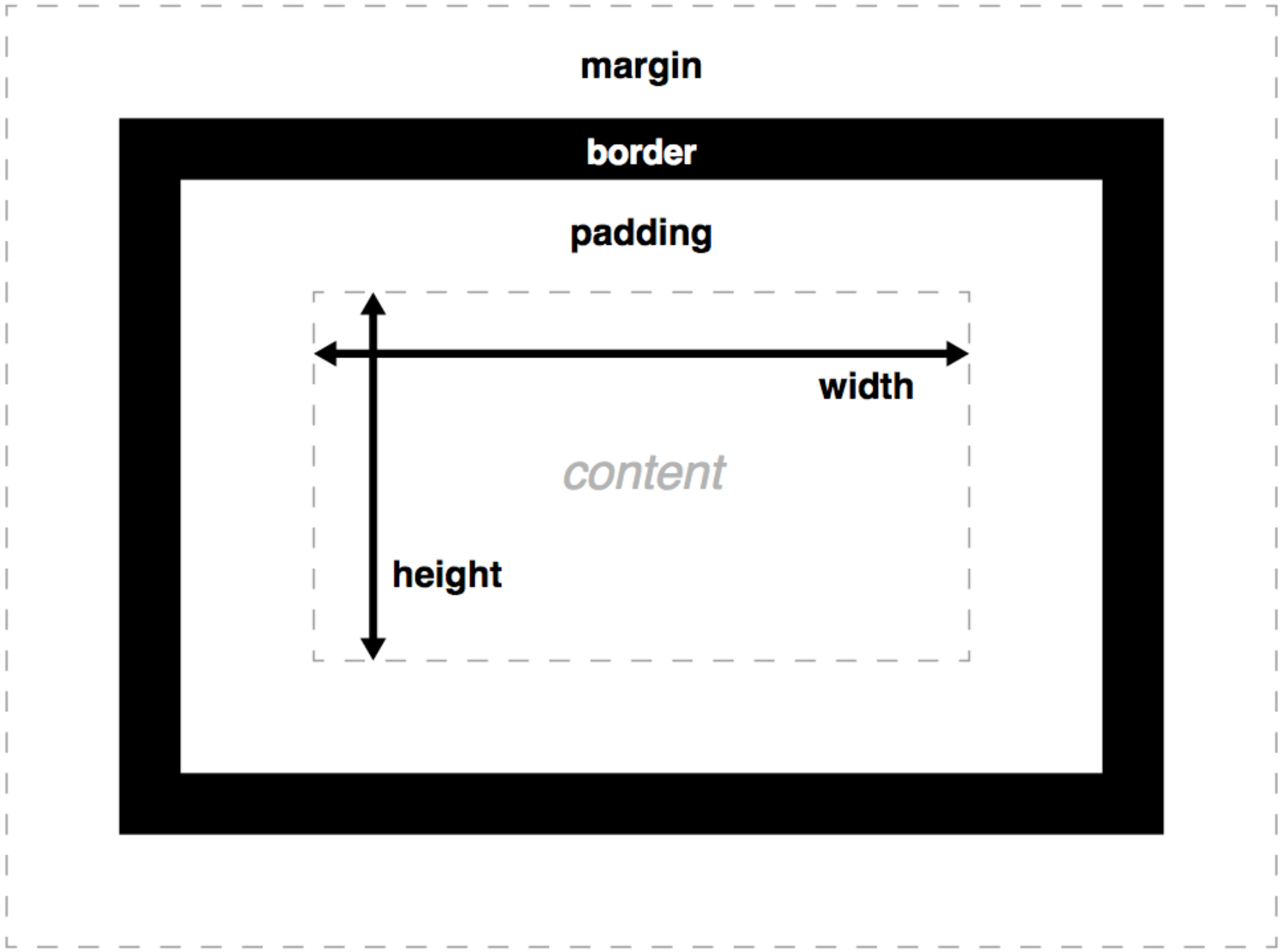
The CSS box model is the foundation of layout on the Web — each element is represented as a rectangular box, with the box's content, padding, border, and margin built up around one another like the layers of an onion. As a browser renders a web page layout, it works out what styles are applied to the content of each box, how big the surrounding onion layers are, and where the boxes sit in relation to one another. Before understanding how to create CSS layouts, you need to understand the box model — this is what we'll look at in this article.

Prerequisites:	Basic computer literacy, basic software installed, basic knowledge of working with files, HTML basics (study

	Introduction to HTML), and an idea of How CSS works (study the previous articles in this module.)
Objective:	To learn how the CSS box model works, and how individual elements are laid out on a page.

Box properties

Every element within a document is structured as a rectangular box inside the document layout, the size and "onion layers" of which can be tweaked using some specific CSS properties. The relevant properties are as follows:



`width` and `height`

The `width` and `height` properties set the width and height of the **content box**, which is the area in which the content of the box is displayed — this content includes both text content set inside the box, and other boxes representing nested child elements.

> **Note**: Other properties exist that allow more subtle ways of handling content box size — setting size constraints rather than an absolute size. This can be done with the properties `min-width`, `max-width`, `min-height`, and `max-height`.

`padding`

Padding refers to the *inner* margin of a CSS box — between the outer edge of the content box and the inner edge of the border. The size of this layer can be set on all four sides at once with the `padding` shorthand property, or one side at a time with the `padding-top`, `padding-right`, `padding-bottom` and `padding-left` properties.

`border`

The **border** of a CSS box sits between the outer edge of the padding and the inner edge of the margin. By default the border has a size of 0 — making it invisible — but you can set the thickness, style and color of the border to make it appear. The `border` shorthand property allows you to set all of these on all four sides at once, for example `border: 1px solid black`. This can be broken down into numerous different longhand properties for more specific styling needs:

- `border-top`, `border-right`, `border-bottom`, `border-left`: Set the thickness, style and color of one side of the border.
- `border-width`, `border-style`, `border-color`: Set only the thickness, style, or color individually, but for all four sides of the border.
- You can also set one of the three properties of a single side of the border individually, using `border-top-width`, `border-top-style`, `border-top-color`, etc.

`margin`

The margin surrounds a CSS box, and pushes up against other CSS boxes in the layout. It behaves rather like `padding`; the shorthand property is `margin` and the individual properties are `margin-top`, `margin-right`, `margin-bottom`, and `margin-left`.

Note: Margins have a specific behavior called margin collapsing: When two boxes touch against one another, the distance between them is the value of the largest of the two touching margins, and not their sum.

Active learning: playing with boxes

At this point, let's jump into an active learning section and carry out some exercises to illustrate some of the particulars of the box model that we discussed above. You can try these exercises in the live editor below, but it might be easier to see some of the effects if you create separate HTML and CSS files locally and try it in a separate browser window instead. You can find the example code on Github.

If you make a mistake, you can always reset it using the *Reset* button. If you get really stuck, press the *Show solution* button to see a potential answer.

In the editable sample below, we have a set of three boxes, all of which contain text content and have been styled to span the whole of the body width. They are represented by `<header>`, `<main>`, and `<footer>` elements in the markup. We'd like you to concentrate on the bottom three CSS rules — the ones that target each box individually — and try the following:

- Have a look at the box model of each individual element on the page by opening up the browser developer tools and clicking on the elements in the DOM inspector. See Discover browser developer tools for help on how to do this. Each browser has a box model viewer that shows exactly what margin, border and padding is applied to each box, how big the content box is, and the total space the element takes up.

- Set some `margin-bottom` on the `<main>` element, say 20px. Now set some `margin-top` on the `<footer>` element, say 15px. Note how the 2nd one of these actions makes no difference to the layout — this shows margin collapsing in action; the smaller margin's effective width is reduced to 0, leaving only the larger margin.
- Set a `margin` of 30px and a `padding` of 30px on every side of the `<main>` element — note how the space around the element (the margin) and the space between the border and the content (the padding) both increase, causing the actual content to take up a smaller amount of space. Again, check this with the browser developer tools.
- Set a larger border on all sides of the `<main>` element, say 40px, and notice how this takes space away from the content rather than the margin or padding. You could do this by setting a complete new set of values for the width, style and color with the `border` property, e.g. `60px dashed red`, but since the properties are already set in a previous rule, you could just set a new `border-width`.
- By default, the content `width` is set to 100% of the available space (after the margin, border, and padding have taken their share) — if you change the browser window width, the boxes will grow and shrink to stay contained inside the example output window. The `height` of the content will default to the height of the content inside it.
- Try setting a new width and height on the `<main>` element — start with say 400px width and 200px height — and observe the effect. You'll notice that the width no longer changes as the browser window is resized.
- Try setting a percentage width on the `<main>` element instead — say 60% width — and observe the effect. You should see that the width now changes again as the browser window is resized. Remove the `<main>` element's `height` setting for now.
- Try setting your `<main>` element's padding and margin to be 5% on all sides, and observe the result. If you use your browser developer tools to look at the width of the example output window and compare that to the width of the margin/padding, you'll see that this 5% means "5% of the containing element's width." So as the size of the example output window increases, so does the padding/margins.

- Margins can accept negative values, which can be used to cause element boxes to overlap. Try setting margin-top: -50px; on the `<main>` element to see the effect.
- Keep experimenting!

HTML Input

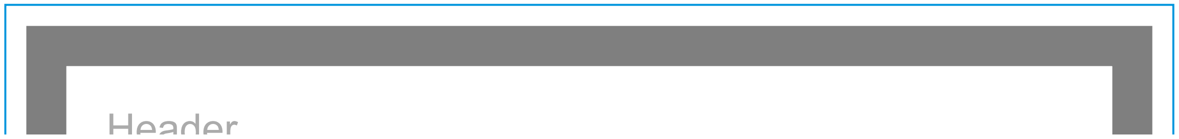
```
<div id="wrapper">
  <header>Header</header>
  <main>Main content</main>
  <footer>Footer</footer>
</div>
```

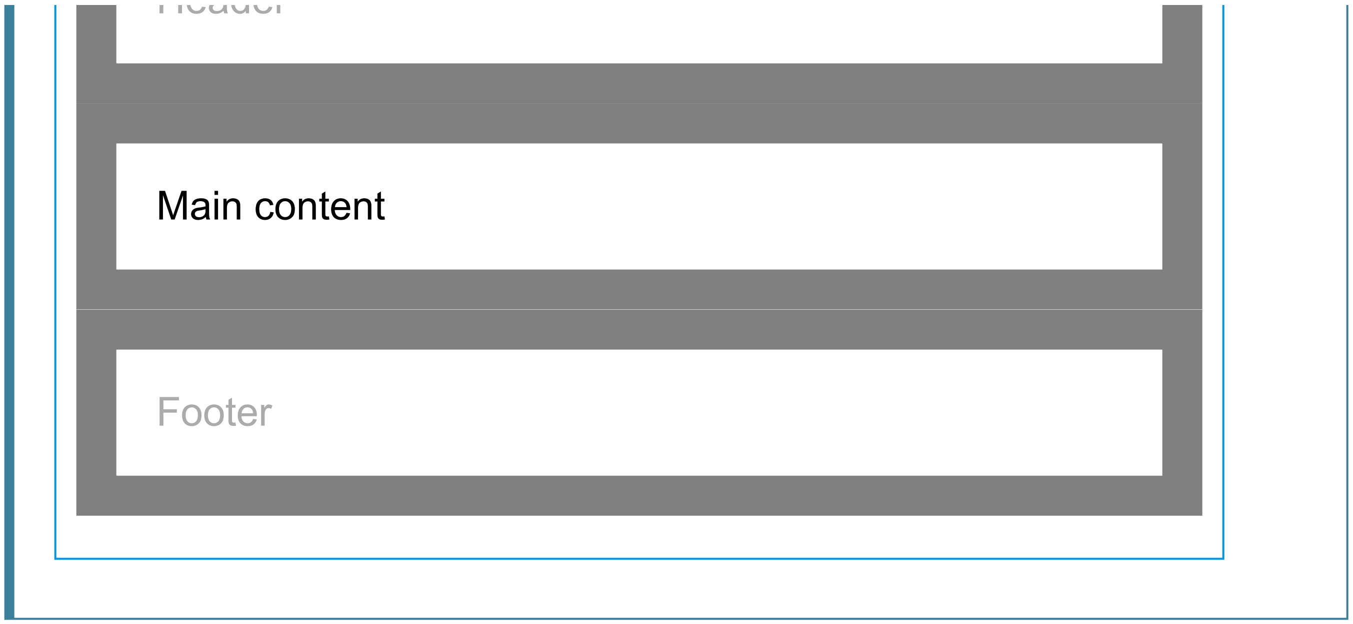
CSS Input

```
/* General styles */
body {
  margin: 0;
}

#wrapper * {
  padding: 20px;
  font-size: 20px;
  border: 20px solid rgba(0,0,0,0.5);
}
```

Output

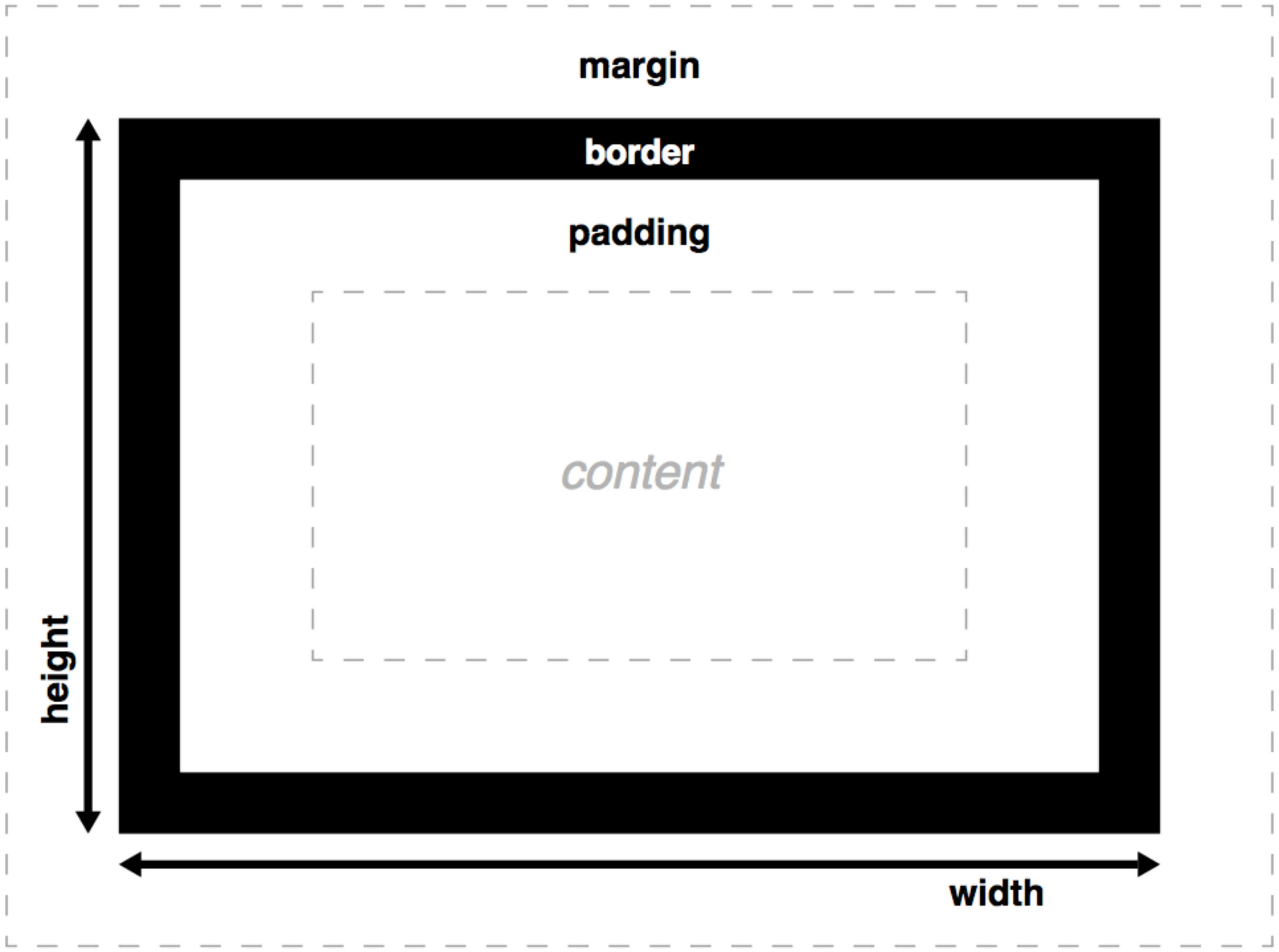




Some hints and ideas:

- By default `background-color`/`background-image` extend to the edge of the border. This behaviour can be changed using the `background-clip` property, which you'll learn about in the Background_clip section.
- If the content box becomes larger than the example output window, it will overflow out of the window, and scroll bars will appear to allow you to scroll the window to view the rest of the box. You can control overflow with the `overflow` property — see also the Overflow section below.
- Box heights don't observe percentage lengths; box height always adopts the height of the box content, unless a specific absolute height is set (e.g. pixels or ems.) This is more convenient than the height of every box on your page defaulting to 100% of the viewport height.
- Borders ignore percentage width settings too.

- You should have noticed that the total width of a box is the sum of its `width`, `padding-right`, `padding-left`, `border-right`, and `border-left` properties. In some cases it is annoying (for example, what if you want to have a box with a total width of 50% with border and padding expressed in pixels?) To avoid such problems, it's possible to tweak the box model with the property `box-sizing`. With the value `border-box`, it changes the box model to this new one:



Advanced box manipulation

Beyond setting the width, height, border, padding and margin of boxes, there are some other properties available to change how they behave. This section discusses those other properties.

Overflow

When you set the size of a box with absolute values (e.g. a fixed pixel width/height), the content may not fit within the allowed size, in which case the content overflows the box. To control what happens in such cases, we can use the `overflow` property. It takes several possible values, but the most common are:

- `auto`: If there is too much content, the overflowing content is hidden and scroll bars are shown to let the user scroll to see all the content.
- `hidden`: If there is too much content, the overflowing content is hidden.
- `visible`: If there is too much content, the overflowing content is shown outside of the box (this is usually the default behavior.)

Here is a simple example to show how these settings work:

First, some HTML:

```html
<p class="autoscroll">
   Lorem ipsum dolor sit amet, consectetur adipiscing elit.
   Mauris tempus turpis id ante mollis dignissim. Nam sed
   dolor non tortor lacinia lobortis id dapibus nunc. Praesent
   iaculis tincidunt augue. Integer efficitur sem eget risus
   cursus, ornare venenatis augue hendrerit. Praesent non elit
   metus. Morbi vel sodales ligula.
```

```html
</p>

<p class="clipped">
   Lorem ipsum dolor sit amet, consectetur adipiscing elit.
   Mauris tempus turpis id ante mollis dignissim. Nam sed
   dolor non tortor lacinia lobortis id dapibus nunc. Praesent
   iaculis tincidunt augue. Integer efficitur sem eget risus
   cursus, ornare venenatis augue hendrerit. Praesent non elit
   metus. Morbi vel sodales ligula.
</p>

<p class="default">
   Lorem ipsum dolor sit amet, consectetur adipiscing elit.
   Mauris tempus turpis id ante mollis dignissim. Nam sed
   dolor non tortor lacinia lobortis id dapibus nunc. Praesent
   iaculis tincidunt augue. Integer efficitur sem eget risus
   cursus, ornare venenatis augue hendrerit. Praesent non elit
   metus. Morbi vel sodales ligula.
</p>
```

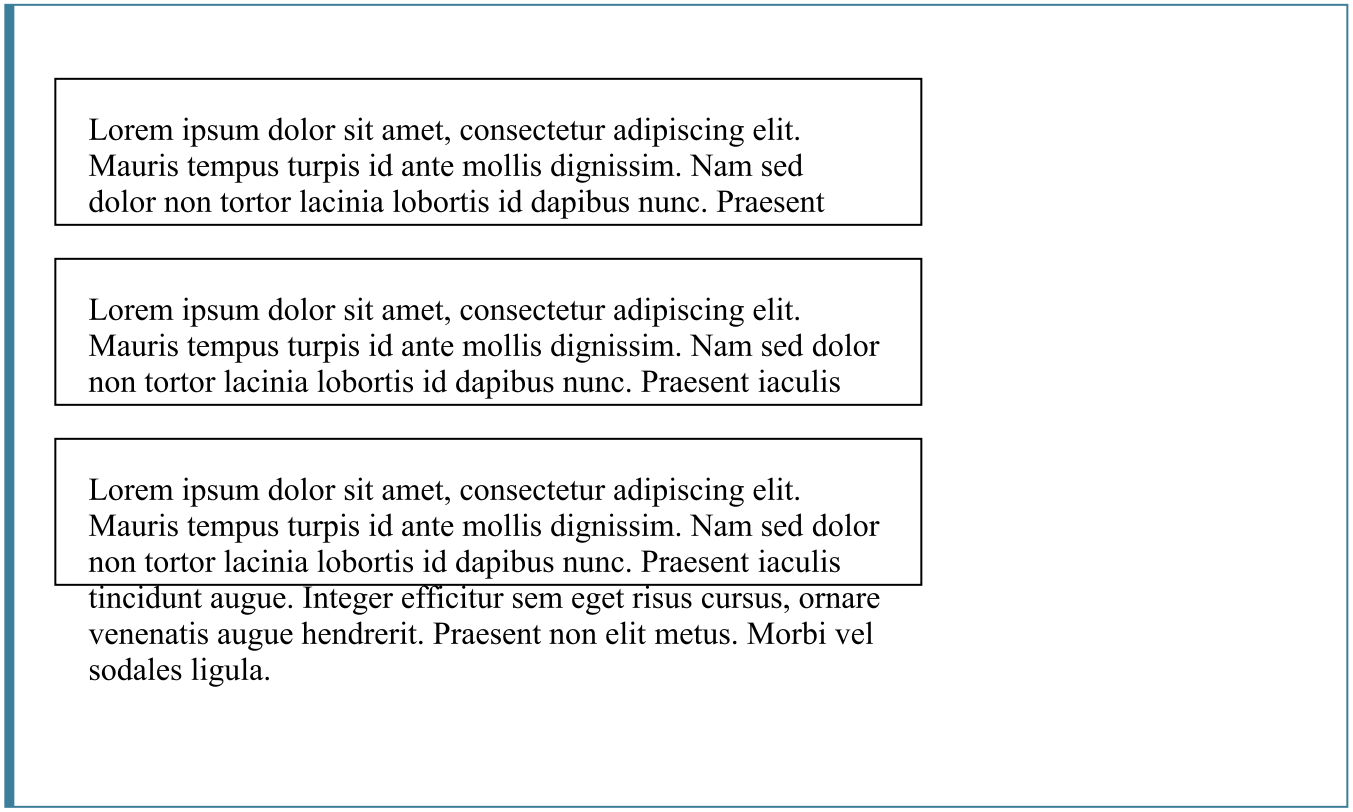
And now some CSS to apply to our HTML:

```css
p {
  width  : 400px;
  height : 2.5em;
  padding: 1em 1em 1em 1em;
  border : 1px solid black;
}

```

```
.autoscroll { overflow: auto;    }
.clipped    { overflow: hidden;  }
.default    { overflow: visible; }
```

The above code gives the following result:



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris tempus turpis id ante mollis dignissim. Nam sed dolor non tortor lacinia lobortis id dapibus nunc. Praesent

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris tempus turpis id ante mollis dignissim. Nam sed dolor non tortor lacinia lobortis id dapibus nunc. Praesent iaculis

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris tempus turpis id ante mollis dignissim. Nam sed dolor non tortor lacinia lobortis id dapibus nunc. Praesent iaculis tincidunt augue. Integer efficitur sem eget risus cursus, ornare venenatis augue hendrerit. Praesent non elit metus. Morbi vel sodales ligula.

Background clip

Box backgrounds are made up of colors and images, stacked on top of each other (`background-color`, `background-image`.) They are applied to a box and drawn under that box. By default, backgrounds extend to the outer edge of the border. This is often fine, but in some cases it can be annoying (what if you have a tiled background image that you want to only extend to the edge of the content?) This behaviour can be adjusted by setting the `background-clip` property on the box.

Let's have a look at an example, to see how this works. First, our HTML:

```html
<div class="default"></div>
<div class="padding-box"></div>
<div class="content-box"></div>
```

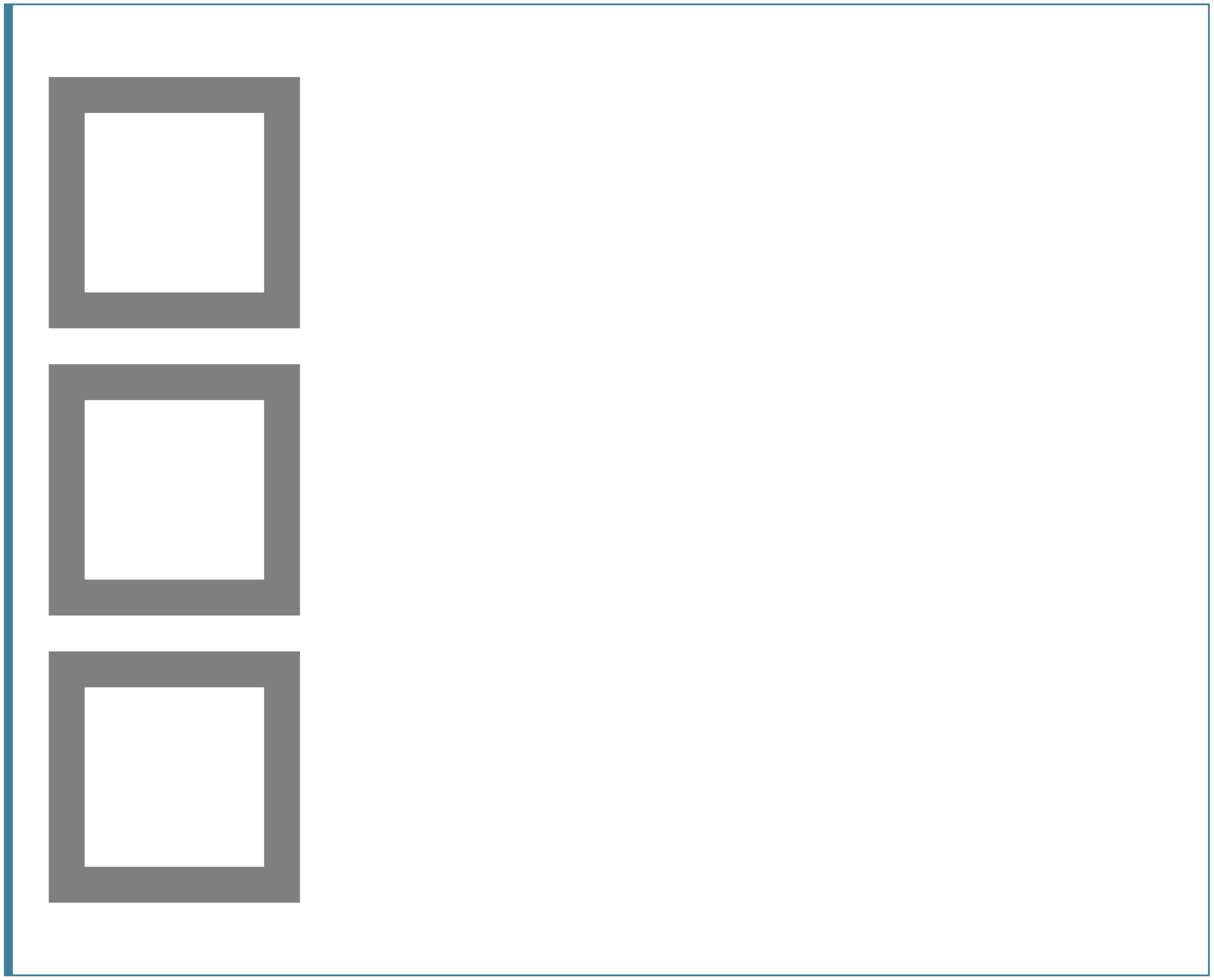
Now the CSS:

```css
div {
  width  : 60px;
  height : 60px;
  border : 20px solid rgba(0, 0, 0, 0.5);
  padding: 20px;
  margin : 20px 0;

  background-size    : 140px;
  background-position: center;
  background-image   : url('https://mdn.mozillademos.org/files/11947/ff-logo.png');
  background-color   : gold;
}

.default     { background-clip: border-box;  }
.padding-box { background-clip: padding-box; }
.content-box { background-clip: content-box; }
```

The above code produces the following result:



Outline

Last but not least, the `outline` of a box is something that looks like a border but which is not part of the box model. It behaves like the border but is drawn on top of the box without changing the size of the box (to be specific, the outline is drawn outside the border box, inside the margin area.)

> **Note**: Beware when using the outline property. It is used in some cases for accessibility reasons to highlight active parts of a web page such as links when a user clicks on them. If you do find a use for outlines, make sure you don't make them look just like link highlights as this could confuse users.

Types of CSS boxes

Everything we've said so far applies to boxes that represent block level elements. However, CSS has other types of boxes that behave differently. The type of box applied to an element is specified by the `display` property. There are many different values available for `display`, but in this article we will focus on the three most common ones; `block`, `inline`, and `inline-block.`

- A `block` box is defined as a box that's stacked upon other boxes (i.e. content before and after the box appears on a separate line), and can have width and height set on it. The whole box model as described above applies to block boxes.
- An `inline` box is the opposite of a block box: it flows with the document's text (i.e. it will appear on the same line as surrounding text and other inline elements, and its content will break with the flow of the text, like lines of text in a paragraph.) Width and height settings have no effect on `inline` boxes; any padding, margin and border set on `inline` boxes will update the position of surrounding text, but will not affect the position of surrounding `block` boxes.

- An `inline-block` box is something in between the first two: It flows with surrounding text and other inline elements without creating line breaks before and after it unlike a `block` box, but it can be sized using width and height and maintains its block integrity like a `block` box. It won't be broken across paragraph lines like an `inline` box. In the below example the `inline-block` box goes onto the 2nd line of text while keeping the shape of a box as there is not enough space for it on the first line, whereas `inline` box *does* break on multiple lines if there is not enough space — it loses the shape of a box.

> **Note**: By default, block level elements have `display: block;` set on them, and inline elements have `display: inline;` set on them.

This may sound a bit confusing at the moment; Let's take a look at a simple example for now.

First, the HTML:

```
<p>
   Lorem ipsum dolor sit amet, consectetur adipiscing elit.
   <span class="inline">Mauris tempus turpis id ante mollis dignissim.</span>
   Nam sed dolor non tortor lacinia lobortis id dapibus nunc.
</p>

<p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit.
  <span class="block">Mauris tempus turpis id ante mollis dignissim.</span>
  Nam sed dolor non tortor lacinia lobortis id dapibus nunc.
</p>

<p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit.
  <span class="inline-block">Mauris tempus turpis id ante mollis dignissim.</span>
  Nam sed dolor non tortor lacinia lobortis id dapibus nunc.
</p>
```

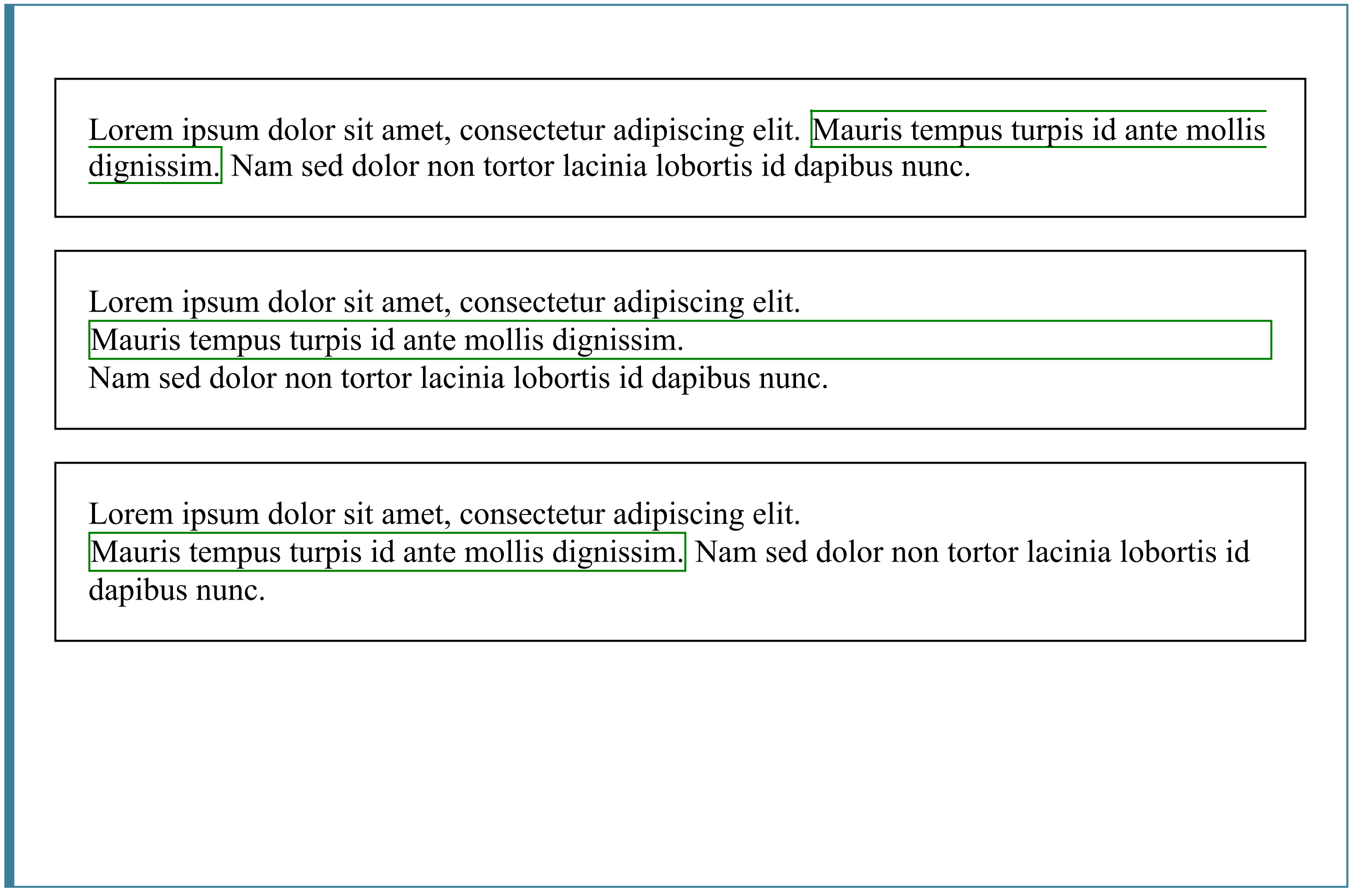
Now let's add some CSS:

```
p {
  padding : 1em;
  border  : 1px solid black;
}

span {
  border  : 1px solid green;
  /* That makes the box visible, regardless of its type */
  background-color: yellow;
}

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

This above code gives this result, which illustrates the different effect of the display types:


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris tempus turpis id ante mollis dignissim. Nam sed dolor non tortor lacinia lobortis id dapibus nunc.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Mauris tempus turpis id ante mollis dignissim.
Nam sed dolor non tortor lacinia lobortis id dapibus nunc.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Mauris tempus turpis id ante mollis dignissim. Nam sed dolor non tortor lacinia lobortis id dapibus nunc.

What's next

At this point, you should have some familiarity with CSS boxes and how they work. Don't worry if you don't fully understand all of this now — you can feel free to reread this article to gain a better understanding, plus you'll start to understand things better when you start to work through some more practical examples later in the course. Next we will have a look at Debugging CSS.

See also

- Block formatting context: The technical term for a CSS box laid out on a web page.
- Visual formatting model: An in depth explanation of the algorithm that lays out CSS boxes on a web page.

Previous   Overview: Introduction to CSS   Next

In this module

- Debugging CSS
- Fundamental CSS comprehension