BOX ALIGNMENT
“As CSS developers, we are programmers of boxes.”
—Lara Schenck
Where do boxes come from?
**Evolution of CSS Specifications**

<table>
<thead>
<tr>
<th>CSS1</th>
<th>CSS2</th>
<th>CSS2.1</th>
<th>CSS2.2</th>
</tr>
</thead>
</table>

- **CSS3**  
  Decision to modularize: 14 Apr 2000  
  (26 modules)

**CSS Snapshot 2017**  
(88 modules)

### Completed
- CSS Snapshots 2017
- CSS Snapshots 2016
- CSS Snapshots 2010
- CSS Snapshots 2007
- CSS Color Level 3
- CSS Class Level 3
- CSS Level 2 Revision 1
- CSS Level 1
- CSS Print Profile
- Media Queries
- CSS Style Attributes

### Stable
- CSS Backgrounds and Borders Level 3
- CSS Conditional Rules Level 3
- CSS Multi-column Layout Level 1
- CSS Values and Units Level 3
- CSS Cascading and Inheritance Level 3
- CSS Fonts Level 3
- CSS Writing Modes Level 3
- CSS Counter Styles Level 3

### Refining
- CSS Animations
- Web Animations 1.0
- CSS Text Level 3
- CSS Transforms
- CSS Transitions
- CSS Box Alignment Level 3
- CSS Display Level 3
- Preview of CSS Level 2
- CSS Timing Functions Level 1

### Testing
- CSS Image Values and Replaced Content Level 3
- CSS Speech
- CSS Flexible Box Layout Level 1
- CSS Text Decoration Level 3
- CSS Shapes Level 1
- CSS Masking Level 1
- CSS Fragmentation Level 3
- CSS Cascading Variables
- Compositing and Blending Level 1
- CSS Syntax Level 3
- CSS Grid Layout Level 1
- CSS Basic User Interface Level 3
- CSS Will Change Level 1
- CSS Media Queries Level 4
- Semantics Interfaces Level 1
- CSS Cascading and Inheritance Level 4
- CSS Scroll Snap Level 1
- CSS Containment Level 1

### Exploring
- CSS Backgrounds and Borders Level 4
- CSS Device Adaptation
- CSS Exclusions
- Filter Effects
- CSS Generated Content for Paired Media
- CSS Page Root
- CSS Template Layout
- CSS Line Grid
- CSS Lists Level 3
- CSS Positioned Layout Level 3
- CSS Regions
- CSS Table Level 3
- CSS Object Model
- CSS Font Loading
- CSS Scoping Level 1
- Non-element Selectors
- CSS Inline Layout Level 3
- Motion Path Level 1
- CSS Blend Display Level 1
- CSS Web User Interface Level 4
- CSS Text Level 4
- CSS Painting API Level 1
- CSS Properties and Values API Level 1
- CSS Typing DOM Level 1
- Worklets Level 1
- CSS Color Level 4
- CSS Font Level 4
- CSS Rhythmic Sizing Level 1
- CSS Image Values and Replaced Content Level 4
- CSS Fill and Stroke Level 3
- CSS Logical Properties and Values Level 1
- CSS Overflow Level 4
“If we hadn’t developed CSS, we could have ended up with the web being a giant fax machine”

— Håkon Wium Lie
CSS1 assumes a simple box-oriented formatting model where each element results in one or more rectangular boxes. (Elements that have a 'display' value of 'none' are not formatted and will therefore not result in a box.) All boxes have a core content area with optional surrounding padding, border and margin areas.
- per pixel control: CSS1 values simplicity over level of control, and although the combination of background images and styled HTML is powerful, control to the pixel level is not possible.
- author control: the author cannot enforce the use of a certain sheet, only suggest
- a layout language: CSS1 does not offer multiple columns with text-flow, overlapping frames etc.

We expect to see extensions of CSS in several directions:
- paper: better support for printing HTML documents
- support for non-visual media: work is in the process to add a list of properties and corresponding values to support speech and braille output
- color names: the currently supported list may be extended
- fonts: more precise font specification systems are expected to complement existing CSS1 font properties.
- values, properties: we expect vendors to propose extensions to the CSS1 set of values and properties. Extending in this direction is trivial for the specification, but interoperability between different UAs is a concern
- layout language: support for two-dimensional layout in the tradition of desktop publishing packages.
- other DTDs: CSS1 has some HTML-specific parts (e.g. the special status of the 'CLASS' and 'ID' attributes) but should easily be extended to apply to other DTDs as well.

We do not expect CSS to evolve into:
- a programming language
9.1 Introduction to the visual formatting model

This chapter and the next describe the visual formatting model: how user agents process the document tree for visual media.

In the visual formatting model, each element in the document tree generates zero or more boxes according to the box model. The layout of these boxes is governed by:

- box dimensions and type.
- positioning scheme (normal flow, float, and absolute positioning).
- relationships between elements in the document tree.
- external information (e.g., viewport size, intrinsic dimensions of images, etc.).

The properties defined in this chapter and the next apply to both continuous media and paged media. However, the meanings of the margin properties vary when applied to paged media (see the page model for details).
Wait, what?
<doctype html>
<html lang="en">
<head>
  <title>CSS rocks</title>
</head>
<body>
  <div>
    <h1>Title</h1>
    <p>Flying rabbits, whoa!</p>
  </div>
... 
... 
</body>
</html>
Box dimensions and type

Positioning scheme
(normal flow / float / absolute positioning)

Layout of boxes

Relationships between elements in the document tree

External information
(e.g. viewport size, intrinsic dimensions of images etc.)
Positioning schemes

- Normal flow
- Floats
- Absolute positioning
What is a formatting context?
The display property

Defines an element’s **display type**, which consists of the two basic qualities of how an element generates boxes.
Inner display type
Defines the generated formatting context for descendant boxes

Outer display type
Dictates a principal box’s own participation in flow layout
<table>
<thead>
<tr>
<th>Short display</th>
<th>Full display</th>
<th>Generated box</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>—</td>
<td>subtree omitted from box tree</td>
</tr>
<tr>
<td>contents</td>
<td>—</td>
<td>element replaced by content in box tree</td>
</tr>
<tr>
<td>block</td>
<td>block flow</td>
<td>block-level block container</td>
</tr>
<tr>
<td>flow-root</td>
<td>block flow-root</td>
<td>block-level block container that establishes a new block formatting context (BFC)</td>
</tr>
<tr>
<td>inline</td>
<td>inline flow</td>
<td>inline box</td>
</tr>
<tr>
<td>inline-block</td>
<td>inline flow-root</td>
<td>inline-level block container</td>
</tr>
<tr>
<td>list-item</td>
<td>block flow list-item</td>
<td>block box with additional marker box</td>
</tr>
<tr>
<td>inline list-item</td>
<td>inline flow list-item</td>
<td>inline box with additional marker box</td>
</tr>
<tr>
<td>run-in</td>
<td>run-in flow</td>
<td>run-in box (inline box with special box-tree-munging rules)</td>
</tr>
<tr>
<td>Short display</td>
<td>Full display</td>
<td>Generated box</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>---------------</td>
</tr>
<tr>
<td>flex</td>
<td>block flex</td>
<td>block-level flex container</td>
</tr>
<tr>
<td>inline-flex</td>
<td>inline flex</td>
<td>inline-level flex container</td>
</tr>
<tr>
<td>grid</td>
<td>block grid</td>
<td>block-level grid container</td>
</tr>
<tr>
<td>inline-grid</td>
<td>inline grid</td>
<td>inline-level grid container</td>
</tr>
<tr>
<td>ruby</td>
<td>inline ruby</td>
<td>inline-level ruby container</td>
</tr>
<tr>
<td>block ruby</td>
<td>block ruby</td>
<td>block box containing ruby container</td>
</tr>
<tr>
<td>table</td>
<td>block table</td>
<td>block-level table wrapper box containing table box</td>
</tr>
<tr>
<td>inline-table</td>
<td>inline table</td>
<td>inline-level table wrapper box containing table box</td>
</tr>
</tbody>
</table>
Block formatting context

The context that block-level boxes participate in

Boxes are laid out one after another, in the block flow direction, from the start of the containing block

Margins along the block flow direction between adjacent block-level boxes in the same block formatting context collapse
Who establishes new block formatting contexts?

- Floats
- Absolutely positioned elements
- Block containers that are not block boxes
- Block boxes with overflow other than visible
- Boxes with display set to flow-root
We need a new BFC because...?

1. Prevent collapsing margins

This is a line of text in a p tag.

I'm a box with margins.

I'm another box with margins.
2. Stop text from flowing around the float

I'm a floated box! This is just a bunch of text that is going on and on so it's long enough to wrap around the float, line boxes yo!

```html
<div class="block-wrapper">
    <div class="box1">I'm a floated box!</div>
    <p class="box2">This is just a bunch of text that is going</p>
</div>
```

```css
.stop-flow .box1 {
    float: left;
}
.stop-flow .box2 {
    display: table-cell;
}
```
3. Contains floats

Floaty! ^_^ Floaty too! :)
Inline formatting contexts

Established by a block container box that contains no block-level boxes

Boxes are laid out one after another, in the inline direction, from the start of the containing block
Inline box construction

If an element <em>generates</em> zero boxes, was it <strong>really there</strong> at all?

```html
<p class="line-container">If an element <em>generates</em> zero boxes</p>
```

```
.linebox .line-container em {
  padding: 0.5em;
  background-color: limegreen;
}

.linebox .line-container strong {
  padding: 0.5em;
  background-color: salmon;
  mix-blend-mode: screen;
}
```
Alignment along the inline-axis

The `text-align` property aligns inline boxes along the inline-axis.

Applicable only when there is extra space available in the line box.
Alignment along the block-axis

Boxes may be aligned along the block-axis in different ways, with the vertical-align property
The height of the line box is based on its font, and its line-height.
Explaining the inline-block centring technique

I'm a block-level box that needs to be centred along the block-axis.

```
.centring .wrapper {
    height: 100%;
}

.centring .wrapper::after {
    content: '\';
    display: inline-block;
    height: 100%;
    vertical-align: middle;
    background-color: limegreen;
    width: 2px;
}

.centring .centred {
    display: inline-block;
    vertical-align: middle;
    width: 50%;
    border: 2px dashed;
}
```

Centering in the Unknown by Chris Coyier
Flex formatting context

Established by a block-level or inline-level `flex` container box
Flex axes

Main axis

Cross axis

horizontal-tb row
Flex lines

nowrap

wrap

wrap-reverse
### Flex directions

```
@directions .wrapper {
  display: flex;
  flex-wrap: wrap;
  writing-mode: horizontal-tb;
  flex-direction: row;
}

@directions .box {
  height: 6em;
  width: 6em;
  border: 1px solid;
}
```

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
All the directions

- LTR
- RTL
- horizontal-tb
- vertical-rl
- vertical-lr
- sideways-lr
- wrap
- wrap-reverse
- row
- row-reverse
- column
- column-reverse

\[ 2 \times 4 \times 2 \times 4 = 64 \]
Aligning with auto margins

```
.automargin{
  display: flex;
}

.automargin div {
  border: 1px solid;
  margin: auto
}
```
Defining “auto” by Elika Etemad (AKA fantasai)
Aligning along the main axis

justify-content helps distribute extra free space left over after flexible lengths and auto margins are resolved.

```
.mainaxis .wrapper {
  display: flex;
  flex-wrap: wrap;
  justify-content: flex-start;
}

.mainaxis .box {
  height: 5em;
  width: 5em;
  border: 1px solid;
}
```
flex-start

flex-end

center

space-between

space-around
Aligning along the cross axis

align-items sets the default alignment for all flex items along the cross axis of the flex line. Over-ridable by align-self.
Packing flex lines

align-content aligns flex lines within the flex container if there is extra space along the **cross-axis**.
Grid formatting context

Established by a block-level or inline-level grid container box
Grid terminology

- Grid line
- Grid track
- Grid area
- Grid cell
- Grid gap
<table>
<thead>
<tr>
<th>Property</th>
<th>Axis</th>
<th>Aligns</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>justify-content</td>
<td>main/inline</td>
<td>content within element (effectively adjusts padding)</td>
<td>block containers, flex containers and grid containers</td>
</tr>
<tr>
<td>align-content</td>
<td>cross/block</td>
<td></td>
<td></td>
</tr>
<tr>
<td>justify-self</td>
<td>inline</td>
<td>element within parent (effectively adjusts margins)</td>
<td>block-level boxes, absolutely-positioned boxes and grid items</td>
</tr>
<tr>
<td>align-self</td>
<td>cross/block</td>
<td></td>
<td>absolutely-positioned boxes, flex items and grid items</td>
</tr>
<tr>
<td>justify-items</td>
<td>inline</td>
<td>items inside box (controls child items)</td>
<td>block containers and grid containers</td>
</tr>
<tr>
<td>align-items</td>
<td>cross/block</td>
<td></td>
<td>flex-containers and grid-containers</td>
</tr>
<tr>
<td>Values</td>
<td>justify-content</td>
<td>align-content</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>center</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>start</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>space-around</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>space-between</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>space-evenly</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Values</td>
<td>justify-content</td>
<td>align-content</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>center</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>start</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>space-around</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>space-between</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>space-evenly</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
justify/align-content

content-distribution properties

```css
.content {
  justify-content: normal;
  align-content: normal;

display: grid;
grid-template-columns: repeat(3, 25%);
grid-template-rows: repeat(3, 20%);
grid-gap: 1em;
grid-template-areas:
  "a b b"
  "a b b"
  "c c d";
}

.content_itemA { grid-area: a }
.content_itemB { grid-area: b }
.content_itemC { grid-area: c }
.content_itemD { grid-area: d }
```
justify/align-self

self-alignment properties

```
.self {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-gap: 1em;
  grid-auto-rows: calc(25% - 1em);
  grid-template-areas:
    "a a b b"
    "a a b b"
    "c c d d"
    "c c d d";
}
.self_itemA {
  grid-area: a;
}
.self_itemB {
  grid-area: b;
}
```
justify/align-items

defaults for justify/align-self

```css
.items {
  justify-items: stretch;
  align-items: stretch;

  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-gap: 1em;
  grid-auto-rows: calc(25% - 1em);
  grid-template-areas:
    "a a b b"
    "a a b b"
    "c c d d"
    "c c d d";
}

.items_itemA {
  grid-area: a;
}
```
Overflow alignment keywords

.A
.B

.C
.D

.overflow {
    justify-content: normal;
    align-content: safe end;

display: grid;
grid-template-columns: repeat(3, 25%);
grid-template-rows: repeat(3, 35%);
grid-gap: 1em;
grid-template-areas:
"a b b"
"a b b"
"c c d";
}

.overflow_itemA { grid-area: a }
.overflow_itemB { grid-area: b }
.overflow_itemC { grid-area: c }
.overflow_itemD { grid-area: d }
Bauhaus in the browser

Featuring...

- Grid
- Flexbox
- Writing mode
- Transforms
- Box alignment

https://codepen.io/huijing/pen/PpqomV | Full page demo
These are not the borders you are looking for

That's more like it
References and resources

- Inside a super fast CSS engine: Quantum CSS (aka Stylo)
- CSS2.2 Visual formatting model
- CSS Display Module Level 3
- CSS2.2 Tables
- CSS Box Alignment Module Level 3
- CSS Writing Modes Level 3
- Bug 1038294 - [css-display] Implement the multi-
  keyword syntax for the 'display' property
- Learn CSS the pedantic way
- Vertical-Align: All You Need To Know
- Understanding CSS Layout And The Block Formatting Context
- The New Layout Standard For The Web: CSS Grid, Flexbox And Box Alignment
- Demystifying CSS alignment
THANK YOU!

https://www.chenhuijing.com

@hj_chen

@hj_chen

@huijing

Font used is Space Text, by Florian Karsten