

CSS



# Objectives

2

- Understand major CSS concepts
- See real life CSS samples
- Present common issues and fix them

# The Big Three

3

- The three major aspects of every web application
  - ▣ Content
  - ▣ Logic
  - ▣ Styling
- Each aspect should be defined separately and independently to allow ease of maintenance
  - ▣ HTML
  - ▣ Java Script
  - ▣ CSS

# CSS

4

- Allow for separation of content from styling
- Markup page content

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title></title>
  </head>
  <body>
    <div id="header"></div>
    <div id="body"></div>
    <div id="footer"></div>
  </body>
</html>
```

- Then apply style to it

```
#header {
  font-size: 2em;
}

#footer {
  border-top: 1px solid black;
  font-size: 0.85em;
}
```

# CSS Versions

5

- CSS 1.0
  - ▣ Visual styles
  - ▣ Layout
- CSS 2.1
  - ▣ Positioning
- CSS 3.0
  - ▣ Transform
  - ▣ Animation
  - ▣ Media Query (View Port)
  - ▣ Much more ...

# Applying Stylesheet

6

- Styles can be applied to a page in different ways
  - ▣ Link
  - ▣ Import
  - ▣ Embedded
  - ▣ In line
- Link is the preferred way
  - ▣ Separation of concerns
  - ▣ Caching

# Linking a Style sheet

7

- Use **link** element

```
<link rel="stylesheet" href="Site.css" type="text/css" />
```

- Browser must download the CSS file before continue rendering the page
  - ▣ Therefore, CSS links are usually located inside the HEAD element
  - ▣ The HTML automatically being rendered with the correct styling
- HTML5 allows you to drop the type attribute

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

# Embedding

8

- Styling instructions are embedded directly inside the HTML
  - Faster HTML rendering since browser does not send another HTTP request

```
<head>
  <title></title>
  <style type="text/css">
    #header {
      font-size: 2em;
    }

    #footer {
      border-top: 1px solid black;
      font-size: 0.85em;
    }
  </style>
</head>
```

- Can remove the type attribute when running under HTML5 compatible browser



# In line Styling

9

- Styling instructions that are located inside an HTML element using the **style** attribute
  - ▣ Should be used rarely
  - ▣ Cannot be overwritten by external styling

```
<body>
  <div id="header" style="font-size: 2em;">Header</div>

  <div id="body"></div>

  <div id="footer" style="border-top: 1px solid black; font-size: 0.85em;"></div>
</body>
```

# Importing

10

- ❑ One stylesheet file may import other stylesheet
- ❑ Nice from maintenance aspect
- ❑ Bad for performance
  - ❑ Browser need to issue another HTTP request

```
@import url("Second.css");

#header {
    font-size: 2em;
}

#footer {
    border-top: 1px solid black;
    font-size: 0.85em;
}
```

# Cascading

11

- Style declarations cascade down to elements from many origins
- The cascade combines
  - ▣ Importance
  - ▣ Origin
  - ▣ Specificity
  - ▣ Order
- Decides which declaration should be applied to any given element

# Style Rules

12

- A rule consist of two parts
  - ▣ Selector
  - ▣ List of styles in the form of property: value;

```
#footer
{
    border-top: 1px solid black;
    font-size: 0.85em;
}
```

- Writing high quality CSS files is all about building the right selector with the right styling

# CSS Selector

13

- Can be seen as a specification on its own
- An expression based language which allows the developer to select one or more elements
- Can be applied to any XML based document
- Don't we have XPath ?
  - ▣ CSS usually performs better
  - ▣ CSS is less powerful
  - ▣ CSS was born before XPath

# Basic Selectors

14

- All tags of particular type

```
button {  
  border: none;  
  padding: 1em;  
}
```

- The universal selector

```
* {  
  font-family: Tahoma;  
}
```

# ID Selector

15

- From HTML perspective **id** attribute must be unique cross all elements
- From CSS perspective this is not a limitation
- ID selector may return more than one element

```
#header {  
    font-size: 2em;  
}
```

- Still, it is not common to set the same ID to different elements
- Thus, we need a different way to treat multiple element in the same way ... See next slide

# Class Selector

16

- HTML element can be associated with zero or more CSS classes

```
<button id="button1" class="btn btn-large">Button 1</button>
```

- All elements with a specified class can be selected using the dot syntax

```
.btn-large {  
    font-size: 2em;  
}
```

- Can select elements with multiple classes

```
.btn.btn-large {  
    font-size: 2em;  
}
```



# Relationship Selectors

17

- Select all links that are descendants of a div

```
div a {  
    color: blue;  
}
```

- Select all links that are direct children of a div

```
div > a {  
    color: blue;  
}
```

- Following sibling (not previous)

```
button + button {  
    color: red;  
}
```

# Sibling Selector

18

- `+` selects only immediate element that is following the left side of the operand
- Below selector does not match the second button

```
button + button {  
  color: red;  
}
```

```
<button>Button 1</button>  
<span></span>  
<button>Button 2</button>
```

- `~` does the magic

```
button ~ button {  
  color: red;  
}
```

- There is no “previous sibling” selector in CSS

# Attribute selectors

19

## □ Select element if a certain attribute is present

- Regardless of whether the attribute has a value

```
<style type="text/css">  
  a[href] {color: blue;}  
</style>
```

## □ Check for an attribute's value

```
<style type="text/css">  
  img[src="spacer.gif"] {width: 5px;}  
</style>
```

## □ Check for an attribute's partial value

- use `~=` for space separated, use `|=` for hyphen-separated

```
<style type="text/css">  
  table[title~= "important"] {width: 100%;}  
</style>
```

# More attribute selectors

20

- Check for an attribute beginning with a value

```
<style type="text/css">  
  table[title^="important"] {width: 100%;}  
</style>
```

- Check for an attribute ending with a value

```
<style type="text/css">  
  table[title$="important"] {width: 100%;}  
</style>
```

- Check for an attribute's partial value

```
<style type="text/css">  
  table[title*="important"] {width: 100%;}  
</style>
```

# Pseudo Class Selectors

21

- Selectors with a colon preceding them
- The most known sample is the A pseudo classes

```
a:visited {  
    color: blue;  
}  
  
a:active {  
    color: green;  
}
```

- Order is important
  - ▣ All pseudo classes are of the same specificity level
  - ▣ Therefore, last wins

# More Pseudo Class Selectors

22

- **first-child**: The first element within a parent
- **first-of-type**: The first element of a specific type
  - ▣ Allows you to select the first IMG inside a DIV even when the IMG is not the first child

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

  <button>Button</button>
</div>
```

```
div button:first-child {
  background-color: blue;
}

div button:first-of-type {
  background-color: blue;
}
```

# Pseudo Element Selectors

23

- Select “unreal” element that does not exist on the page
- Are used to inject content without modifying the HTML ... Didn't we say separation of concerns?
- **::before** – Creates a pseudo-element that is the first child of the element matched
- **::after**
- **::first-letter**
- **::first-line**

# CSS 2 vs. CSS 3

24

- CSS 2
  - ▣ Both pseudo class and pseudo element use colon “:”
- CSS 3
  - ▣ Pseudo class uses colon “:”
  - ▣ Pseudo element uses double colon “::”
  - ▣ Motivation is to distinguish the twos
- Most browser support both



# Injecting Icons using CSS

25

- Is an icon content or styling ?
- Using pseudo-element we can add an icon without modifying the HTML

```
<ul>
  <li class="css">
    <span>Ori</span>
  </li>
  <li class="html">
    <span>Roni</span>
  </li>
</ul>
```

```
li:before {
  position: relative;
  top: 5px;
}

li.css:before {
  content: url(CSS3.ico);
}

li.html:before {
  content: url(HTML.ico);
}
```

# Specificity

26

- Assuming the following markup

```
<div>  
  <button class="btn">Button</button>  
</div>
```

- And the following selectors

```
div button {  
  color: red;  
}  
  
.btn {  
  color: green;  
}
```

- Who wins ?

# Specificity – Formal Definition

27

- Count the number of ID selectors (=a)
- Count the number of class selectors, attribute selectors, pseudo-class (=b)
- Count the number of type selectors and pseudo-element (=c)
- Ignore the universal selector
- Selectors inside negation are counted as usual. The negation itself does not count
- Concatenate the three number a-b-c
  - ▣ Assuming a number system with a large base

# Specificity - Samples

28

□ `div button {  
    color: red;  
}` → 2

□ `.btn {  
    color: green;  
}` → 10

□ `#header .progressbar a {  
    text-decoration: none;  
}` → 111

- Inline style is always stronger
- In case of two selectors with same specificity the later wins

# !important

29

- Style rule annotated with **!important** is more specific than any other selectors
  - ▣ Even inline rule 😊
- Is considered bad practice
  - ▣ Hard to maintain

```
<button style="color: red;">Button 1</button>
```

```
button {  
    color: blue !important;  
}  
  
button {  
    color: green !important;  
}
```

- In case of multiple !important styles → Later wins

# Why a selector is ignored ?

30

- Consider following Markup

```
<body>  
  <p>  
    <div>Hello</div>  
  </p>  
</body>
```

- And the following rules

```
p div {  
  font-size: 2em;  
}
```

- What is wrong ?

# Invalid HTML

31

- According to the HTML specification P tag can only contain inline elements
- When browser encounters DIV inside P it assumes that the DIV element is a sibling and therefore “closes” the P tag

```
<body>  
  <p></p>  
  <div>Hello</div>  
  <p></p>  
</body>
```

- The following rule DOES match

```
p + div {  
  font-size: 2em;  
}
```

# Inheritance

32

- Some styles are inherited from parent element
  - ▣ font-size, color, background-color
- Other styles can be set by the designer to be inherited

```
.child {  
  background-color: blue;  
  padding: inherit;  
}
```

- The inheritance rule is weak
  - ▣ Every new definition overrides it



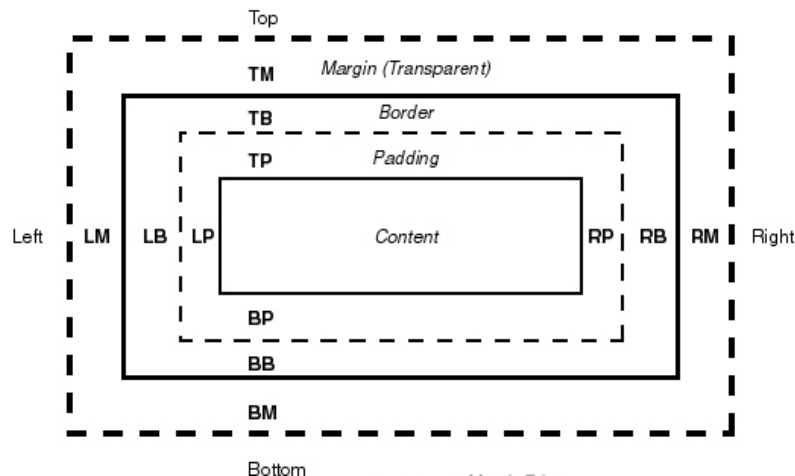
# The CSS box model

33

□ Every element is treated as a box

□ A box consists of

- ▣ Content
- ▣ Padding
- ▣ Border
- ▣ Margin



□ Margin vs. padding

- ▣ The margin takes the colour of the container's background
- ▣ The padding takes the colour of the content's background

# CSS Box model

34

- The total width that an element occupies is the addition of content, padding and border
- Setting element's width using CSS only effect the content width, not the total width
- Usually, this behavior breaks our design
  - ▣ See next slide

# Was IE right ?

35

- Suppose we have two child elements that are exactly filling their parent element

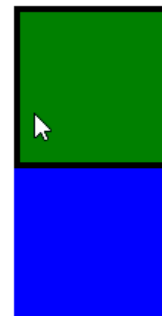
```
<div class="parent">  
  <div class="child child1"></div>  
  <div class="child child2"></div>  
</div>
```

```
.child1 {  
  background-color: green;  
}  
  
.child2 {  
  background-color: blue;  
}
```



- We want to add border when mouse is hovering
  - ▣ The extra border pushes the second element to the next line

```
.child:hover {  
  border: 2px solid black;  
}
```

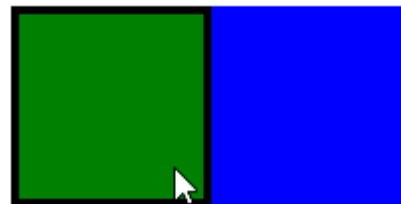


# box-sizing

36

- By default CSS width means content width
- **box-sizing** changes the box model
- When box-sizing is **border-box** the CSS width means content+padding+border

```
.child {  
  width: 100px;  
  height: 100px;  
  float: left;  
  box-sizing: border-box;  
}
```



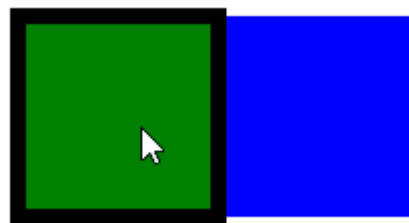
- Other supported values: content-box, padding-box

# Outline

37

- Continuing our previous example
- It would be nicer if the border appears around the element and not inside it
  - ▣ Thus element remains the same size
  - ▣ Sibling elements should not be effected

```
.child:hover {  
  outline: 4px solid black;  
  position: relative;  
}
```



- Why position relative ?

# Colliding Margin

38

- ❑ Two plus two doesn't always equal four
- ❑ When bottom margin of one element touches the top margin of another
- ❑ The browser applies the larger of the two

```
<div class="block1"></div>  
<div class="block2"></div>
```



```
.block1 {  
  height: 1em;  
  background-color:red;  
  margin-bottom: 1em;  
}  
  
.block2 {  
  height: 1em;  
  background-color:blue;  
  margin-top: 1em;  
}
```

# Go with the flow

39

- HTML tags act much like text in a word-processing program
  - ▣ Tags are filling the entire width of a page and flowing from top to bottom
- Each box flows one after the other
  - ▣ Except special boxes
- Many problems can be solved by flowing with the flow ...

# Inline vs. Block element

40

- Not all boxes are alike
- Two different types
  - ▣ Block
  - ▣ Inline
- Block element creates a break before and after it
  - ▣ For example, P tag
- Inline element appears on the same line as the content and tags beside them



# Inline Element's Padding & Margin

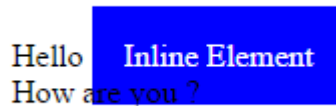
41

## □ Assuming following markup & CSS

```
<body>
  Hello <span class="label">Inline Element</span>
  <br /> How are you ?
</body>
```

```
.label {
  color: white;
  background-color: blue;
  padding: 1em;
}
```

## □ The result is surprising



Hello    Inline Element  
How are you ?

# Inline Element's Padding & Margin

42

- ❑ Vertical padding and margin have no effect on sibling elements
- ❑ Horizontal padding and margin are honored
- ❑ You can change the element display style to **inline-block**

```
.label {  
  color: white;  
  background-color: blue;  
  padding: 1em;  
  display: inline-block;  
}
```

Hello **Inline Element**  
How are you ?

# Inline Element's Height & Width

43

- Height and width are completely ignored !!!

```
<body>
  Hello <span class="label">Inline Element</span><br />
  How are you ?
</body>
```

```
.label {
  color: white;
  background-color: blue;
  width: 200px;
  height: 200px;
}
```

- Width and height are determined according to the element's content

Hello **Inline Element**  
How are you ?

- Solution: Again, move to **inline-block**

# inline-block Element

44

- From parent perspective inline-block element is just like a plain inline element
  - ▣ Flows from left to right
  - ▣ The margin & padding are taken into account
- However, the element itself feels like a block
  - ▣ By default width and height are set according to content
  - ▣ Or you can change them to other values

# Line Box

45

- Inline elements are arranged inside a virtual line space
- CSS defines a set of rules to determine the position of an element inside the line
  - ▣ Horizontal positioning is straightforward
  - ▣ Vertical positioning is tricky

# Line's Baseline

46

- By default all inline element are positioned according to the line's baseline

```
<div class="line">
  <span class="label1">&#xC4g</span>
</div>
```

```
.line {
  background-color: green;
  width: 4em;
}

.label1 {
  background-color: red;
  font-size: 0.5em;
}
```



# Where is the baseline ?

47

- This is the tricky part
- Officially, the line's baseline is the position where a non styled “x” (lowercase) is located
- However, there are many factors that effect the position of the baseline
  - ▣ Font
  - ▣ Minimization of the line height
  - ▣ Elements inside the line ☹

# vertical\_align

48

- Effects only inline elements
- Determines the vertical position of an element relative to the line
- Supports
  - ▣ baseline (default)
  - ▣ bottom/top
  - ▣ middle
  - ▣ sub/super
  - ▣ text-top/text-bottom



# Centering an Icon

49

- You want to attach a small icon with a line of text

```
<div class="line">  
  <span class="icon"></span>  
  <span class="label1">Very very long text ...</span>  
</div>
```

■ ABCDEFGHIJKLMOP

- The icon is not vertically centered
- Let's use `vertical_align`

```
.icon {  
  ...  
  vertical-align: middle;  
}
```

■ ABCDEFGHIJKLMOP

- Oops ... worse

# vertical\_align middle

50

- middle is relative to the line's baseline ☹️
  - ▣ Not to the line vertical center point
- How can we change the location of the baseline ?
  - ▣ The trick is to define an element that its height equals to the line's height
  - ▣ Then set its vertical align property to middle
  - ▣ Since the element cannot be moved inside the line (too tall) the browser moves the baseline (Wow ...)

# vertical\_align middle

51

```
.icon {  
  display: inline-block;  
  height: 16px;  
  width: 16px;  
  background-color: red;  
  vertical-align: middle;  
}  
  
.label1 {  
  font-size: 1em;  
  vertical-align: middle;  
}
```

■ ABCDEFGHIJKLMOP

- Most designers are surprised that changing **.label1** `vertical_align` property actually effects the **.icon** position

# Vertical Centering

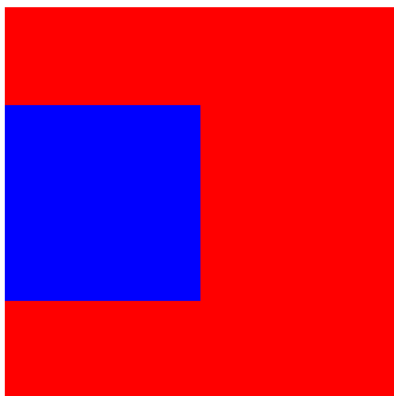
52

- Can the `vertical_align` be used to implement a general vertical centering of an element ?
- Only if
  - ▣ Parent consists of one line
  - ▣ Child is inline element
  - ▣ The line's baseline is located at the center point
- The last requirement is difficult
  - ▣ However, we can inject a pseudo element with 100% and `vertical_align middle`

# Vertical Centering

53

```
<div class="parent">  
  <div class="child"></div>  
</div>
```



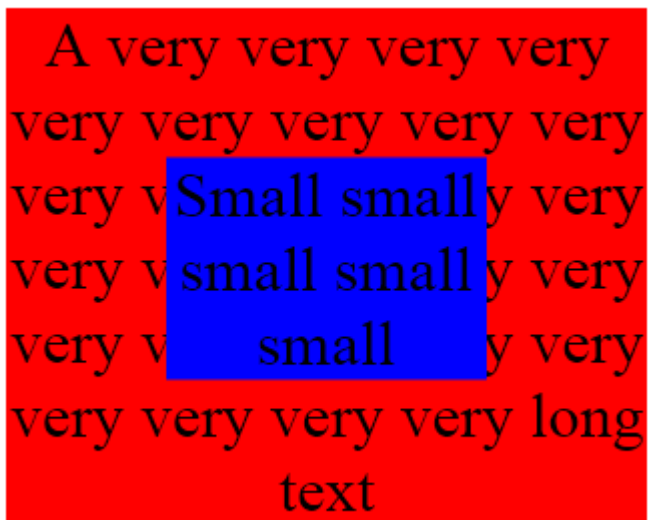
```
.parent {  
  display: inline-block;  
  background-color: red;  
}  
  
.parent:after {  
  content: "";  
  height: 100%;  
  display: inline-block;  
  vertical-align: middle;  
}  
  
.child {  
  display: inline-block;  
  background-color: blue;  
  vertical-align: middle;  
}
```

- Works even if parent and child heights are of unknown value

# Vertical Centering

54

- What if parent consists of two or more lines ?
- Use transform (CSS3)



```
.parent {  
  width: 10em;  
  display: inline-block;  
  background-color: red;  
  position: relative;  
  text-align: center;  
}  
  
.child {  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
  width: 5em;  
  display: inline-block;  
  background-color: blue;  
}
```

# Wrap

55

- Inline element is treated as a single word
- When a line has no more space, the next element is pushed to a new line
- New line might overflow its container (vertically)

```
<div class="parent">  
  Text text text text text text text  
</div>
```

```
.parent {  
  display: inline-block;  
  width: 7em;  
  height: 2em;  
  background-color: red;  
}
```

Text text text  
text text text text  
text

# white-space: nowrap

56

- You can prevent element wrapping by setting **white-space** to **nowrap**

```
<div class="parent">  
  Text text text text text text text text  
</div>
```

```
.parent {  
  display: inline-block;  
  width: 7em;  
  height: 2em;  
  background-color: red;  
  white-space: nowrap;  
}
```

Text text text text text text text text

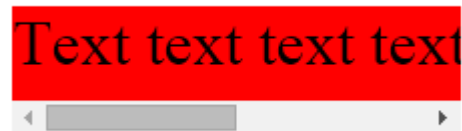


# Controlling the overflow

57

- By default the overflow content is visible
- Can configure parent's **overflow** style to
  - **hidden** - Hide the overflow content
  - **scroll** - Always display a scrollbar
  - **auto** - Display a scrollbar only if overflow exists

```
.parent {  
  display: inline-block;  
  width: 7em;  
  height: 2em;  
  background-color: red;  
  white-space: nowrap;  
  overflow-x: auto;  
}
```



# Exact filling

58

- Suppose we have two child elements that need to **exactly** fill their parent

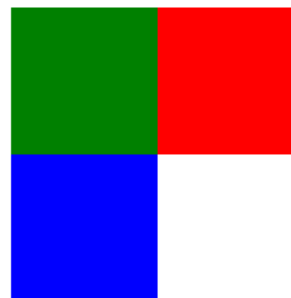
```
<div class="parent">  
  <div class="child1"></div>  
  <div class="child2"></div>  
</div>
```

```
.parent {  
  background-color: red;  
  width: 200px;  
  height: 200px;  
}
```

```
.child1 {  
  background-color: green;  
  width: 100px;  
  height: 200px;  
}
```

```
.child2 {  
  background-color: blue;  
  width: 100px;  
  height: 200px;  
}
```

- The result is surprising

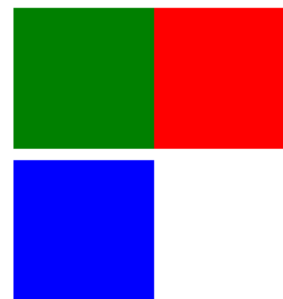


# Block means block

59

- A block element breaks the line
- Following element are moved to next line
- Setting a width for the block element has no effect on the line breaking
- Change child DIVs to inline-block

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



- Ooops ... why is that ?

# White Space

60

- There is no difference between one white space and multiple white space characters
- However, one white space character does occupy some space (0.25em)

```
<div class="parent">  
  <div class="child1"></div><div class="child2"></div>  
</div>
```

```
<div class="parent">  
  <div class="child1"></div><!--  
  --><div class="child2">  
    </div>  
</div>
```

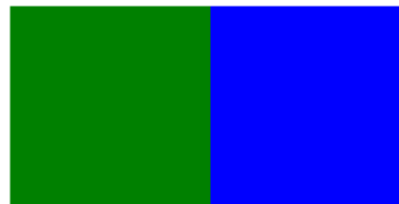


# Zero font-size

61

- Previous problem can be solved by zeroing the font-size
- However, remember that font-size is inherited to child elements
- Therefore, you may need to “restore” the font-size for child elements

```
.parent {  
  background-color: red;  
  width: 200px;  
  height: 100px;  
  font-size: 0;  
}
```

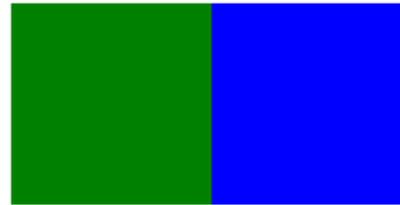


# Floating

62

- Floating can solve our problem too

```
.child {  
  width: 100px;  
  height: 100px;  
  float: left;  
}
```



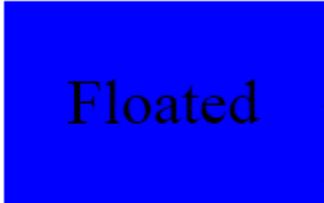
- However, float has side effects that must be considered
  - ▣ See next slides for more details

# Floating – Float First

63

- Floated element must come before the content itself

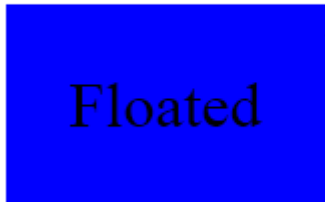
```
<div class="parent">
  <span class="floated">Floated</span>
  Text text text text text text
</div>
```



The diagram shows a blue rectangular box labeled "Floated" on the left. To its right, the text "Text text" is on the top line, "text text text" is on the middle line, and "text text" is on the bottom line. The text is aligned to the right of the floated box.

- Not following the rule creates strange layout

```
<div class="parent">
  Text text text text text text
  <span class="floated">Floated</span>
</div>
```



The diagram shows the text "Text text text text text" on the top line and "text text" on the bottom line. Below the text, there is a blue rectangular box labeled "Floated". The box is positioned below the text it is supposed to float, creating a strange layout.

# Floating – Collapsing Height

64

- A block parent with only floated children has no height

```
<div class="parent">
  <span class="floated">floated</span>
  <span class="floated">floated</span>
</div>
```

```
.floated
{
  float: left;
}
```

```
.parent
{
  background-color: red;
}
```

floatedfloated

- Can fix that with overflow: hidden

```
.parent
{
  background-color: red;
  overflow: hidden;
}
```

**floatedfloated**



# Floating – Border and Background

65

- Text wraps around floating elements
- Border and background do not

```
<div class="view">
  <div class="floated">
    Floated floated floated
  </div>
  <div class="header">
    Header header header header header
  </div>
  <div class="title">
    Title
  </div>
</div>
```

Header header header  
header header  
Title

Floated  
floated  
floated

- Can fix that with `overflow: hidden` (again ...)

```
.header {
  overflow: hidden;
}

.title {
  overflow: hidden;
}
```

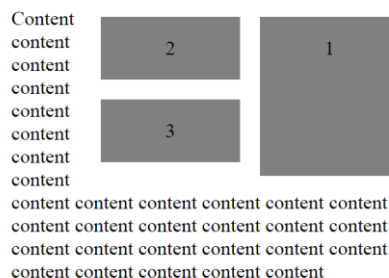
Header header header  
header header  
Title

Floated  
floated  
floated

# Stopping the Float

66

- Floating elements sit one after the other (horizontal ordering)



- What if we want the floating elements to sit one below the other

```
.floated
{
  clear: both;
}
```

Content content content  
 content content content  
 content content content  
 content content content  
 content content content  
 content content content  
 content content content  
 content content content  
 content content content  
 content content content  
 content content content



# CSS Positioning

67

- CSS offers four types of positioning
  - ▣ Static – The default behavior
  - ▣ Absolute
  - ▣ Relative
  - ▣ Fixed
- Remember our moto: “Go with the flow”

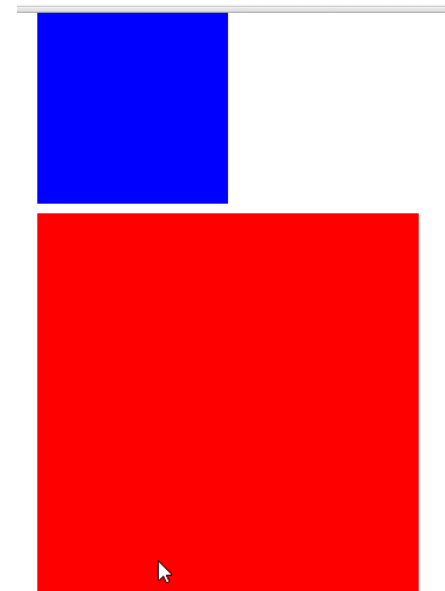
# Absolute Positioning

68

- Position an element relative to the boundaries of its closest positioned ancestor
  - ▣ Positioned ancestor = parent element with non default position style

```
<div class="parent">  
  <div class="child">  
  </div>  
</div>
```

```
.child {  
  height: 5em;  
  width: 5em;  
  background-color: blue;  
  position: absolute;  
  top: 0;  
}
```



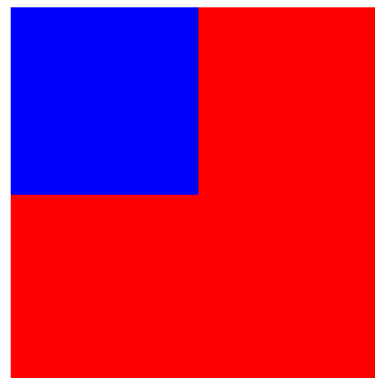
- The .child element has no positioned ancestor and therefore is positioned relative to the window

# Absolute Positioning

69

- Common scenario is to position an element relatively to its parent while keeping the parent in the exact location
- Solution: Use **relative** positioning

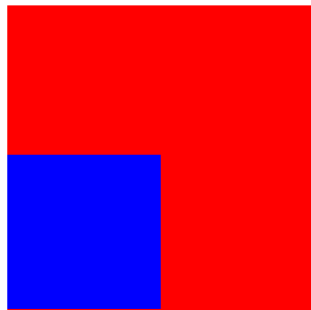
```
.parent {  
  height: 10em;  
  width: 10em;  
  background-color: red;  
  position: relative;  
}
```



# Absolute Position: bottom & right

70

- Absolute positioning using top and left is straightforward
- However, what does it mean absolute position with bottom equal 5px ?
- Answer: The bottom edge of the child element should be located 5px above the bottom edge of the parent

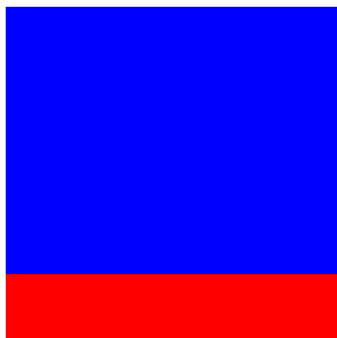


# Absolute Position: Docking

71

- Absolute positioning can be used to dock an element
- For example, bottom docking

```
<div class="parent">  
  <div class="child dock-bottom">  
  </div>  
</div>
```



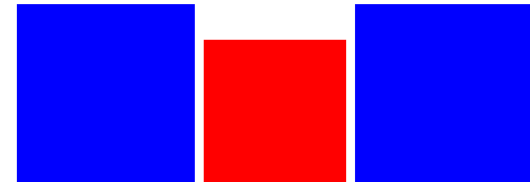
```
.parent {  
  width: 10em;  
  height: 10em;  
  background-color: blue;  
  position: relative;  
}  
  
.child {  
  height: 2em;  
  background-color: red;  
}  
  
.dock-bottom {  
  position: absolute;  
  left: 0;  
  right: 0;  
  bottom: 0;  
}
```

# Absolute Position: Out of flow

72

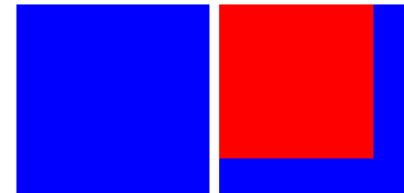
- Absolute positioned element is out of the normal document flow
- As if all other elements don't know that the element exist

```
<div class="child1 child"></div>  
<div class="child2 child"></div>  
<div class="child3 child"> </div>
```



- Now, lets add position: absolute

```
.child2 {  
  background-color: red;  
  position: absolute;  
}
```



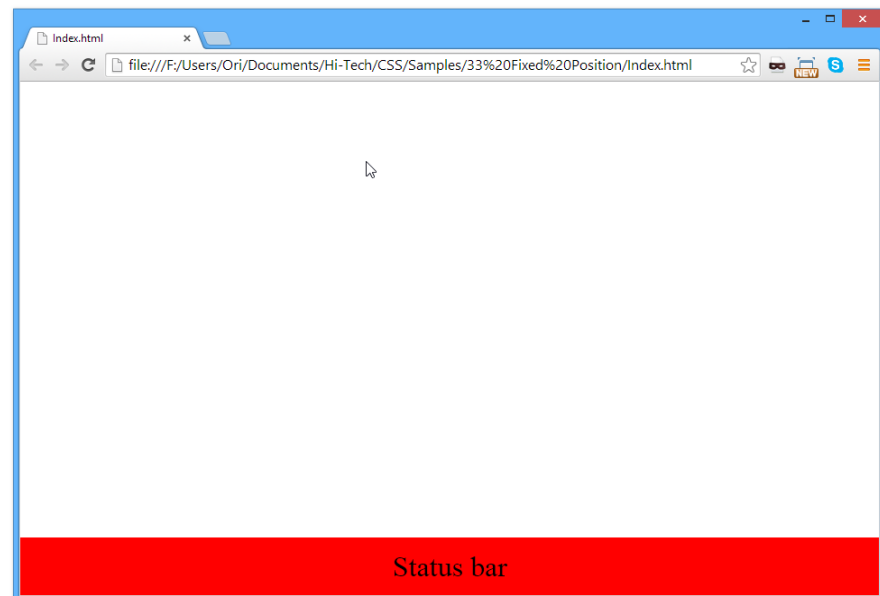


# Fixed Position

73

- Position an element relative to the boundaries of the window
- Beside the different parent it behaves the same as absolute positioning
  - Out of flow

```
.status-bar {  
    position:fixed;  
    height: 2em;  
    line-height: 2em;  
    background-color: red;  
    left:0;  
    right:0;  
    bottom:0;  
    text-align: center;  
}
```

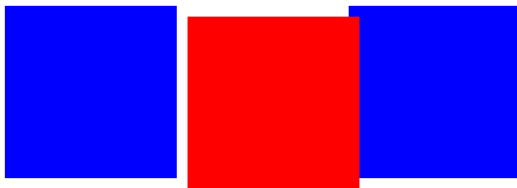


# Relative Position

74

- Position an element relative to itself
  - ▣ Are you serious ?
- Actually, very useful
  - ▣ Relative positioned element + `left: 5px` → move 5px to the right while keeping all other elements untouched

```
<div class="child1 child"></div>  
<div class="child2 child"></div>  
<div class="child3 child"></div>
```



```
.child {  
  width: 5em;  
  height: 5em;  
  background-color: blue;  
  display: inline-block;  
  float: left;  
}  
  
.child2 {  
  background-color: red;  
  position: relative;  
  left: 10px;  
  top: 10px;  
}
```

# z-index

75

- Absolute positioned elements might overflow each other
- By default, the later absolute positioned element has higher z-index priority

```
<div class="child child1"></div>  
<div class="child child2"></div>
```



```
.child {  
  position: absolute;  
  width: 5em;  
  height: 5em;  
}  
  
.child1 {  
  top: 1em;  
  left: 1em;  
  background-color: red;  
}  
  
.child2 {  
  top: 2em;  
  left: 2em;  
  background-color: green;  
}
```

# Size Units

76

## □ So many options

Type		Type	
px	Pixel	pc	12pt
in	Inch	ex	Relative to “x”
cm	Centimeter	ch	Relative to “0”
mm	Millimeter	vw/vh/vmin/vmax	Relative to view port
em	Relative to font-size	%	Like em
rem	Root em		
pt	1/72 inch		

## □ You probably need only three: em, rem, px

# The Pixel unit - px

77

- px is easily understood
- Parent size has no effect
- Browser zoom may effect
- Be aware, that 1 CSS pixel does not necessary means 1 hardware pixel
  - ▣ Apple Retina and other modern monitors
- Most of the time using px is not convenient
  - ▣ You probably want to change parent size and accept all children to adjust accordingly

# The M unit - EM

78

- EM means the size of the M letter (printing world)
- Inside the CSS world 1em means the size of the base font
  - ▣ By default is 16px (most browsers)
- Changing font family has no effect
- em is a relative unit
  - ▣ Changing parent's font-size effects children

# em vs. px

79

## □ 1em = 16px

```
<div class="child child1">Child1</div>  
<div class="child child2">Child2</div>
```

```
.child1 {  
    font-size: 2em;  
}  
  
.child2 {  
    font-size: 32px;  
}
```

Child1  
Child2

## □ When changing parent font-size

```
html  
{  
    font-size: 32px;  
}
```

Child1  
Child2

# The Root EM - rem

80

- Size is relative to the HTML tag's font-size and not to the parent

```
<div class="parent">  
  Parent  
  <div class="child1">  
    Child1  
  </div>  
  <div class="child2">  
    Child2  
  </div>  
</div>
```

```
.parent {  
  font-size: 2em;  
}  
  
.child1 {  
  font-size: 2em;  
}  
  
.child2 {  
  font-size: 2rem;  
}
```

Parent  
Child1  
Child2



# Which unit is the best ?

81

- Depends (like always ...)
- Most of the time you should use **em**
- Use **rem** for padding & margin
- Use **px** for border

# Summary

82

- ❑ Mastering CSS is not easy
- ❑ Web Designer is a profession
- ❑ Can greatly improve your skills by understading some major CSS concepts
  - ▣ Box model
  - ▣ Inline vs. block element
  - ▣ Line box
  - ▣ Floating
  - ▣ Unit size