Responsive CSS

CS 390 – Web Application Development

J. Setpal

September 6, 2023

Callout: Machine Learning @ Purdue

If you can view this screen, I am making a mistake.

Outline

1. Why It's Worth Your Time
2. Some Static Stuff
3. Animations
4. ETC

Outline

1. Why It's Worth Your Time
2. Some Static Stuff
3. Animations
4. ETC

WIWYT – Responsive CSS

- Every single website needs to generalize to <u>every device specification</u>.

WIWYT – Responsive CSS

- Every single website needs to generalize to every device specification.
- Animations and Transitions make websites look very cool.

WIWYT – Responsive CSS

- Every single website needs to generalize to <u>every device specification</u>.
- Animations and Transitions make websites look very cool.
- We **want to write less JavaScript**.

Outline

1. Why It's Worth Your Time
2. Some Static Stuff
3. Animations
4. ETC

Positioning Elements

Complicated webpages have a lot of elements interacting simultaneously.

Positioning Elements

Complicated webpages have a lot of elements interacting simultaneously.
By manipulating `position` in CSS, we can update element placement.
Syntax: `target { position: relative|absolute|sticky|fixed }`

Positioning Elements

Complicated webpages have a lot of elements interacting simultaneously.
By manipulating `position` in CSS, we can update element placement.
Syntax: `target { position: relative|absolute|sticky|fixed }`

Each of these values have the following behaviors:

- `static`: Default value; positions elements based on the DOM tree

Positioning Elements

Complicated webpages have a lot of elements interacting simultaneously.
By manipulating `position` in CSS, we can update element placement.
Syntax: `target { position: relative|absolute|sticky|fixed }`

Each of these values have the following behaviors:

- `static`: Default value; positions elements based on the DOM tree
- `relative`: Positions elements relative to the parent element

Positioning Elements

Complicated webpages have a lot of elements interacting simultaneously.
By manipulating `position` in CSS, we can update element placement.
Syntax: `target { position: relative|absolute|sticky|fixed }`

Each of these values have the following behaviors:

- `static`: Default value; positions elements based on the DOM tree
- `relative`: Positions elements relative to the parent element
- `absolute`: Positions elements relative to the HTML document

Positioning Elements

Complicated webpages have a lot of elements interacting simultaneously. By manipulating `position` in CSS, we can update element placement.
Syntax: `target { position: relative|absolute|sticky|fixed }`

Each of these values have the following behaviors:

- `static`: Default value; positions elements based on the DOM tree
- `relative`: Positions elements relative to the parent element
- `absolute`: Positions elements relative to the HTML document
- `sticky`: Element scrolls with it's parent till the parent ends and stays at the viewport edge after

Positioning Elements

Complicated webpages have a lot of elements interacting simultaneously. By manipulating `position` in CSS, we can update element placement.
Syntax: `target { position: relative|absolute|sticky|fixed }`

Each of these values have the following behaviors:

- `static`: Default value; positions elements based on the DOM tree
- `relative`: Positions elements relative to the parent element
- `absolute`: Positions elements relative to the HTML document
- `sticky`: Element scrolls with it's parent till the parent ends and stays at the viewport edge after
- `fixed`: Positions elements relative to the viewport

Alignment – Flexbox

Flexbox is a one dimensional layout method[1] for content alignment within a parent container.

[1] MDN

Alignment – Flexbox

Flexbox is a one dimensional layout method[1] for content alignment within a parent container.

It is triggered by setting `display: flex` in the parent container. It aligns elements on the basis of two **axes**:

- **Main Axis:** Along the axis of alignment.

[1] MDN

Alignment – Flexbox

Flexbox is a one dimensional layout method[1] for content alignment within a parent container.

It is triggered by setting `display: flex` in the parent container. It aligns elements on the basis of two **axes**:

- **Main Axis:** Along the axis of alignment.
- **Cross Axis:** Orthogonal to the axis of alignment.

[1] MDN

Alignment – Flexbox

Flexbox is a one dimensional layout method[1] for content alignment within a parent container.

It is triggered by setting `display: flex` in the parent container. It aligns elements on the basis of two **axes**:

- **Main Axis:** Along the axis of alignment.
- **Cross Axis:** Orthogonal to the axis of alignment.

Parent Attributes:

Key	Use-case
`flex-direction`	row / column main axis

[1] MDN

Alignment – Flexbox

Flexbox is a one dimensional layout method[1] for content alignment within a parent container.

It is triggered by setting `display: flex` in the parent container. It aligns elements on the basis of two **axes**:

- **Main Axis:** Along the axis of alignment.
- **Cross Axis:** Orthogonal to the axis of alignment.

Parent Attributes:

Key	**Use-case**
`flex-direction`	row / column main axis
`flex-wrap`	[don't] move objects to the below line

[1] MDN

Alignment – Flexbox

Flexbox is a one dimensional layout method[1] for content alignment within a parent container.

It is triggered by setting `display: flex` in the parent container. It aligns elements on the basis of two **axes**:

- **Main Axis:** Along the axis of alignment.
- **Cross Axis:** Orthogonal to the axis of alignment.

Parent Attributes:

Key	**Use-case**
`flex-direction`	row / column main axis
`flex-wrap`	[don't] move objects to the below line
`justify-content`	Item spacing against the main axis

[1] MDN

Alignment – Flexbox

Flexbox is a one dimensional layout method[1] for content alignment within a parent container.

It is triggered by setting `display: flex` in the parent container. It aligns elements on the basis of two **axes**:

- **Main Axis:** Along the axis of alignment.
- **Cross Axis:** Orthogonal to the axis of alignment.

Parent Attributes:

Key	Use-case
`flex-direction`	row / column main axis
`flex-wrap`	[don't] move objects to the below line
`justify-content`	Item spacing against the main axis
`align-content`	Item spacing against the cross axis

[1] MDN

Alignment – Flexbox

Flexbox is a one dimensional layout method[1] for content alignment within a parent container.

It is triggered by setting `display: flex` in the parent container. It aligns elements on the basis of two **axes**:

- **Main Axis:** Along the axis of alignment.
- **Cross Axis:** Orthogonal to the axis of alignment.

Parent Attributes:

Key	Use-case
`flex-direction`	row / column main axis
`flex-wrap`	[don't] move objects to the below line
`justify-content`	Item spacing against the main axis
`align-content`	Item spacing against the cross axis
`align-items`	Orientation within child elements

[1] MDN

Flexbox – Child Elements

Child Elements:

Key	**Use-case**
`flex-shrink`	Ratio for space reduction

Flexbox – Child Elements

Child Elements:

Key	**Use-case**
`flex-shrink`	Ratio for space reduction
`flex-grow`	Ratio for space extension

Flexbox – Child Elements

Child Elements:

Key	Use-case
`flex-shrink`	Ratio for space reduction
`flex-grow`	Ratio for space extension
`flex-basis`	Initial item size

Flexbox – Child Elements

Child Elements:

Key	**Use-case**
`flex-shrink`	Ratio for space reduction
`flex-grow`	Ratio for space extension
`flex-basis`	Initial item size
`align-self`	Override child orientation

Compounding Rules

On Monday (August 28, 2023), we spoke about D.R.Y: Don't Repeat Yourself.

[2] w.r.t DOM

Compounding Rules

On Monday (August 28, 2023), we spoke about D.R.Y: Don't Repeat Yourself. Having multiple CSS targets with the same rule can introduce such a situation.

[2] w.r.t DOM

Compounding Rules

On Monday (August 28, 2023), we spoke about D.R.Y: Don't Repeat Yourself. Having multiple CSS targets with the same rule can introduce such a situation.

One way to avoid this is to use **compounded rules**.

[2] w.r.t DOM

Compounding Rules

On Monday (August 28, 2023), we spoke about D.R.Y: Don't Repeat Yourself. Having multiple CSS targets with the same rule can introduce such a situation.

One way to avoid this is to use **compounded rules**.
Syntax: `target1, target2 { k: v; }`

[2] w.r.t DOM

Compounding Rules

On Monday (August 28, 2023), we spoke about D.R.Y: Don't Repeat Yourself. Having multiple CSS targets with the same rule can introduce such a situation.

One way to avoid this is to use **compounded rules**.
Syntax: `target1, target2 { k: v; }`

You can also compound rules to **subset** targets. This targets descendants of specific elements.

[2] w.r.t DOM

Compounding Rules

On Monday (August 28, 2023), we spoke about D.R.Y: Don't Repeat Yourself. Having multiple CSS targets with the same rule can introduce such a situation.

One way to avoid this is to use **compounded rules**.
Syntax: `target1, target2 { k: v; }`

You can also compound rules to **subset** targets. This targets descendants of specific elements.
Syntax: `target1target2 { k: v; }`
English: Apply to all `target2`'s that are descendants[2] of `target1`'s.

[2] w.r.t DOM

Compounded Specificity

$\mathbf{Q_0}$: Recap: What is specificity?

Compounded Specificity

$\mathbf{Q_0}$: Recap: What is specificity?

$\mathbf{Q_1}$: How would you resolve specificity for rules:

A. `ul#primary-nav li.active`, and

B. `nav a:hover::before`?

Compounded Specificity

$\mathbf{Q_0}$: Recap: What is specificity?
$\mathbf{Q_1}$: How would you resolve specificity for rules:

A. `ul#primary-nav li.active`, and
B. `nav a:hover::before`?

$\mathbf{A_1}$: Create a new system (power $\propto$ specificity):

Compounded Specificity

$\mathbf{Q_0}$: Recap: What is specificity?
$\mathbf{Q_1}$: How would you resolve specificity for rules:

A. `ul#primary-nav li.active`, and
B. `nav a:hover::before`?

$\mathbf{A_1}$: Create a new system (power $\propto$ specificity):

1. IDs have power $10^{\mathbf{2}}$.
2. Classes, attributes, pseudo-classes have power $10^{\mathbf{1}}$.
3. Elements, pseudo-elements have power $10^{\mathbf{0}}$.

Compounded Specificity

$\mathbf{Q_0}$: Recap: What is specificity?
$\mathbf{Q_1}$: How would you resolve specificity for rules:

A. `ul#primary-nav li.active`, and
B. `nav a:hover::before`?

$\mathbf{A_1}$: Create a new system (power $\propto$ specificity):

1. IDs have power $10^{\mathbf{2}}$.
2. Classes, attributes, pseudo-classes have power $10^{\mathbf{1}}$.
3. Elements, pseudo-elements have power $10^{\mathbf{0}}$.

Then add the scores; the higher score retains precedence.

Compounded Specificity

$\mathbf{Q_0}$: Recap: What is specificity?
$\mathbf{Q_1}$: How would you resolve specificity for rules:

A. `ul#primary-nav li.active`, and
B. `nav a:hover::before`?

$\mathbf{A_1}$: Create a new system (power $\propto$ specificity):

1. IDs have power $10^{\mathbf{2}}$.
2. Classes, attributes, pseudo-classes have power $10^{\mathbf{1}}$.
3. Elements, pseudo-elements have power $10^{\mathbf{0}}$.

Then add the scores; the higher score retains precedence.

Fantastic reference: `https://specificity.keegan.st/`

CSS Functions

Certain values for CSS elements cannot be precomputed. To enable this use-case, CSS includes a set of predefined **value functions**.

Syntax: `func(*args)`

CSS Functions

Certain values for CSS elements cannot be precomputed. To enable this use-case, CSS includes a set of predefined **value functions**.
Syntax: `func(*args)`

Key	**Use-case**
`var`	Get custom properties

CSS Functions

Certain values for CSS elements cannot be precomputed. To enable this use-case, CSS includes a set of predefined **value functions**.

Syntax: `func(*args)`

Key	**Use-case**
`var`	Get custom properties
`translate<X>`	Compute an updated relative position

CSS Functions

Certain values for CSS elements cannot be precomputed. To enable this use-case, CSS includes a set of predefined **value functions**.

Syntax: `func(*args)`

Key	Use-case
`var`	Get custom properties
`translate<X>`	Compute an updated relative position
`img`	Load an image

CSS Functions

Certain values for CSS elements cannot be precomputed. To enable this use-case, CSS includes a set of predefined **value functions**.

Syntax: `func(*args)`

Key	**Use-case**
`var`	Get custom properties
`translate<X>`	Compute an updated relative position
`img`	Load an image
`rotate<X>`	Compute an updated relative rotation

CSS Functions

Certain values for CSS elements cannot be precomputed. To enable this use-case, CSS includes a set of predefined **value functions**.
Syntax: `func(*args)`

Key	**Use-case**
`var`	Get custom properties
`translate<X>`	Compute an updated relative position
`img`	Load an image
`rotate<X>`	Compute an updated relative rotation
`max`	Compute the maximum of a series of values

CSS Functions

Certain values for CSS elements cannot be precomputed. To enable this use-case, CSS includes a set of predefined **value functions**.

Syntax: `func(*args)`

Key	**Use-case**
`var`	Get custom properties
`translate<X>`	Compute an updated relative position
`img`	Load an image
`rotate<X>`	Compute an updated relative rotation
`max`	Compute the maximum of a series of values
`min`	Compute the minimum of a series of values

CSS Functions

Certain values for CSS elements cannot be precomputed. To enable this use-case, CSS includes a set of predefined **value functions**.

Syntax: `func(*args)`

Key	Use-case
`var`	Get custom properties
`translate<X>`	Compute an updated relative position
`img`	Load an image
`rotate<X>`	Compute an updated relative rotation
`max`	Compute the maximum of a series of values
`min`	Compute the minimum of a series of values
`clamp`	Setup a minimum, ideal and maximum value set

CSS Functions

Certain values for CSS elements cannot be precomputed. To enable this use-case, CSS includes a set of predefined **value functions**.

Syntax: `func(*args)`

Key	Use-case
`var`	Get custom properties
`translate<X>`	Compute an updated relative position
`img`	Load an image
`rotate<X>`	Compute an updated relative rotation
`max`	Compute the maximum of a series of values
`min`	Compute the minimum of a series of values
`clamp`	Setup a minimum, ideal and maximum value set
`calc`	Perform arithmetic operations

@ Rules

At-Rules are descriptive / conditionally applied properties, that inform custom behaviours of the webpage.

@ Rules

At-Rules are descriptive / conditionally applied properties, that inform custom behaviours of the webpage.

@ rules are defined in two ways:

- **Regular Syntax:** `@identifier (RULE);`

@ Rules

At-Rules are descriptive / conditionally applied properties, that inform custom behaviours of the webpage.

@ rules are defined in two ways:

- **Regular Syntax:** `@identifier (RULE);`
- **Nested Syntax:** `@identifier (RULE) { k:v };`

@ Rules

At-Rules are descriptive / conditionally applied properties, that inform custom behaviours of the webpage.

@ rules are defined in two ways:

- **Regular Syntax:** `@identifier (RULE);`
- **Nested Syntax:** `@identifier (RULE) { k:v };`

Key	Type	Use-case
`@charset`	Regular	Define character encoding

@ Rules

At-Rules are descriptive / conditionally applied properties, that inform custom behaviours of the webpage.

@ rules are defined in two ways:

- **Regular Syntax:** `@identifier (RULE);`
- **Nested Syntax:** `@identifier (RULE) { k:v };`

Key	Type	Use-case
`@charset`	Regular	Define character encoding
`@supports`	Nested	Apply if property is supported

@ Rules

At-Rules are descriptive / conditionally applied properties, that inform custom behaviours of the webpage.

@ rules are defined in two ways:

- **Regular Syntax:** `@identifier (RULE);`
- **Nested Syntax:** `@identifier (RULE) { k:v };`

Key	**Type**	**Use-case**
`@charset`	Regular	Define character encoding
`@supports`	Nested	Apply if property is supported
`@media`	Nested	Apply viewport specific rules

@ Rules

At-Rules are descriptive / conditionally applied properties, that inform custom behaviours of the webpage.

@ rules are defined in two ways:

- **Regular Syntax:** `@identifier (RULE);`
- **Nested Syntax:** `@identifier (RULE) { k:v };`

Key	**Type**	**Use-case**
`@charset`	Regular	Define character encoding
`@supports`	Nested	Apply if property is supported
`@media`	Nested	Apply viewport specific rules
`@keyframes`	Nested	Keyframes

@ Rules

At-Rules are descriptive / conditionally applied properties, that inform custom behaviours of the webpage.

@ rules are defined in two ways:

- **Regular Syntax:** `@identifier (RULE);`
- **Nested Syntax:** `@identifier (RULE) { k:v };`

Key	Type	Use-case
`@charset`	Regular	Define character encoding
`@supports`	Nested	Apply if property is supported
`@media`	Nested	Apply viewport specific rules
`@keyframes`	Nested	Keyframes
`@import`	Regular	Use remote CSS styles

Variables in CSS

CSS also allows us to define custom properties, or **variables**.
Syntax: `--property-name: value;`

Variables in CSS

CSS also allows us to define custom properties, or **variables**.
Syntax: `--property-name: value;`

These are accessible only within their defined attribute by default.
Common best practice is to make it accessible globally:
`:root { --property-name: value; }` using the `root` psuedo-class.

Variables in CSS (Contd.)

These values can then be accessed using the `var(--property-name)` function. **Example:**

style.css

```
:root {
        --background-color: blue;
}

h1 {
        background-color: var(--background-color);
}

h2 {
        background-color: var(--background-color);
}
```

Outline

Transition / Animation Dichotomy

Q: Any obvious difference between **transitions** and **animations** (semantic interpretation)?

Transition / Animation Dichotomy

Q: Any obvious difference between **transitions** and **animations** (semantic interpretation)?
A: That's pretty much it! (did the gambit work?)

Transition / Animation Dichotomy

Q: Any obvious difference between **transitions** and **animations** (semantic interpretation)?
A: That's pretty much it! (did the gambit work?)

More formally, **transitions** handle smooth state changes of updating properties.

Transition / Animation Dichotomy

Q: Any obvious difference between **transitions** and **animations** (semantic interpretation)?
A: That's pretty much it! (did the gambit work?)

More formally, **transitions** handle smooth state changes of updating properties.

Animations allow more fine-grained control using keyframes to define the state of the update, and do not need to be triggered by updates within properties.

Transitions

Transitions are generally used when state changes are performed *interactively* – i.e. through the GUI.

Transitions

Transitions are generally used when state changes are performed *interactively* – i.e. through the GUI.

Syntax: The following primary properties define transitions:

1. `transition-property: all|<property-name>`

Transitions

Transitions are generally used when state changes are performed *interactively* – i.e. through the GUI.

Syntax: The following primary properties define transitions:

1. `transition-property: all|<property-name>`
2. `transition-duration: n unit`

Transitions

Transitions are generally used when state changes are performed *interactively* – i.e. through the GUI.

Syntax: The following primary properties define transitions:

1. transition-property: all|<property-name>
2. transition-duration: n unit
3. transition-timing-function: ease|ease-in|ease-in-out|linear

Transitions

Transitions are generally used when state changes are performed *interactively* – i.e. through the GUI.

Syntax: The following primary properties define transitions:

1. `transition-property: all|<property-name>`
2. `transition-duration: n unit`
3. `transition-timing-function: ease|ease-in|ease-in-out|linear`
4. `transition-delay: n unit`

Transitions

Transitions are generally used when state changes are performed *interactively* – i.e. through the GUI.

Syntax: The following primary properties define transitions:

1. `transition-property: all|<property-name>`
2. `transition-duration: n unit`
3. `transition-timing-function: ease|ease-in|ease-in-out|linear`
4. `transition-delay: n unit`

Shorthand: Use `transition` with the arguments in the above order.

Transitions

Transitions are generally used when state changes are performed *interactively* – i.e. through the GUI.

Syntax: The following primary properties define transitions:

1. `transition-property: all|<property-name>`
2. `transition-duration: n unit`
3. `transition-timing-function: ease|ease-in|ease-in-out|linear`
4. `transition-delay: n unit`

Shorthand: Use `transition` with the arguments in the above order.

Multiple transitions can be defined by using comma-separation.

Transitions

Transitions are generally used when state changes are performed *interactively* – i.e. through the GUI.

Syntax: The following primary properties define transitions:

1. `transition-property: all|<property-name>`
2. `transition-duration: n unit`
3. `transition-timing-function: ease|ease-in|ease-in-out|linear`
4. `transition-delay: n unit`

Shorthand: Use `transition` with the arguments in the above order.

Multiple transitions can be defined by using comma-separation. Unlike fonts, these are applied <u>in parallel</u>; not as fallback.

Animations

Animations allow us fine-grained access of the scene, using **keyframes** for controlling the manipulation.
Syntax:

style.css

```
@keyframes <name>
    {
    from|0% {
        k: v;
    }

    to|100% {
        k: v;
    }
}
```

Animations (Contd.)

Within the target we specify the following:

1. `animation-name: <name>;`

Animations (Contd.)

Within the target we specify the following:

1. `animation-name: <name>;`
2. `animation-duration: n unit;`

Animations (Contd.)

Within the target we specify the following:

1. `animation-name: <name>;`
2. `animation-duration: n unit;`
3. `animation-delay: n unit;`

Animations (Contd.)

Within the target we specify the following:

1. `animation-name: <name>;`
2. `animation-duration: n unit;`
3. `animation-delay: n unit;`
4. `animation-iteration-count: infinite;`

Animations (Contd.)

Within the target we specify the following:

1. `animation-name: <name>;`
2. `animation-duration: n unit;`
3. `animation-delay: n unit;`
4. `animation-iteration-count: infinite;`
5. `animation-timing-function: ease[-in][-out]|linear;`

Animations (Contd.)

Within the target we specify the following:

1. `animation-name: <name>;`
2. `animation-duration: n unit;`
3. `animation-delay: n unit;`
4. `animation-iteration-count: infinite;`
5. `animation-timing-function: ease[-in][-out]|linear;`
6. `animation-direction: reverse;`

Animations (Contd.)

Within the target we specify the following:

1. `animation-name: <name>;`
2. `animation-duration: n unit;`
3. `animation-delay: n unit;`
4. `animation-iteration-count: infinite;`
5. `animation-timing-function: ease[-in][-out]|linear;`
6. `animation-direction: reverse;`
7. `animation-fill-mode: none;`

Animations (Contd.)

Within the target we specify the following:

1. `animation-name: <name>;`
2. `animation-duration: n unit;`
3. `animation-delay: n unit;`
4. `animation-iteration-count: infinite;`
5. `animation-timing-function: ease[-in][-out]|linear;`
6. `animation-direction: reverse;`
7. `animation-fill-mode: none;`

Shorthand: Use `animation` with the arguments in the above order.

Outline

1. Why It's Worth Your Time
2. Some Static Stuff
3. Animations
4. ETC

Thank you!

Have an awesome rest of your day!

Slides: `https://cs.purdue.edu/homes/jsetpal/slides/r-css.pdf`

If anything's incorrect or unclear, please ping `jsetpal@purdue.edu`
I'll patch it ASAP.