{POWER.CODERS}

Recap Day

Agenda

- Recap: HTML & CSS
- Theory: Responsive Web Design
- Recap: Pseudo-Code & Algorithms

Recap: HTML & CSS

Mobile first

- in flex
- in grid

Code along

Do it yourself exercise (45 min)

Columns

Do it yourself exercise (45 min)

Simple Layout

Live coding exercise (45 min)

Navigation

Live coding exercise (30 min)

Slider

Media query exercise (60 min)

Style this form and make it responsive

THEORY

Responsive by default

`flex`, `grid` and `multicol` are responsive by default.

Responsive by default

`flex`, `grid` and `multicol` are responsive by default.

Using one or more of these layout techniques will make your website reponsive already.

Media queries

CSS module that enables content to adapt to screen resolutions and sizes.

MEDIA QUERIES

CSS module that enables content to adapt to screen resolutions and sizes.

A media query consists of a test, followed by as many CSS rules as we want, with the rule block enclosed in a set of braces. If the test condition is false, the browser simply ignores the rule block and moves on.

MEDIA QUERIES

CSS module that enables content to adapt to screen resolutions and sizes.

A media query consists of a test, followed by as many CSS rules as we want, with the rule block enclosed in a set of braces. If the test condition is false, the browser simply ignores the rule block and moves on.

A media query can be used to check for a particular condition such as the width and height (of the browser window), device width and height, orientation or resolution.

Example

```
@media screen and (min-width: 800px) { … }
```

Example

```
@media screen and (min-width: 800px) { … }
```

- **Media Type** (example: screen) declares what type of of media the rules should be applied to. There are four options you can declare: all, print, screen, and speech.

Example

```
@media screen and (min-width: 800px) { … }
```

- **Media Type** (example: screen) declares what type of of media the rules should be applied to. There are four options you can declare: all, print, screen, and speech.
- **Expressions** (example: (min-width: 800px) targets devices based on specific conditions, like min-width, max-width, device-pixel-ratio and more.

Example

```
@media screen and (min-width: 800px) { … }
```

- **Media Type** (example: screen) declares what type of of media the rules should be applied to. There are four options you can declare: all, print, screen, and speech.
- **Expressions** (example: (min-width: 800px) targets devices based on specific conditions, like min-width, max-width, device-pixel-ratio and more.
- **CSS Rules** are defined inside the curly brackets and only apply when media tpye and expressions both return True.

BREAKPOINTS

Media queries are used to define breakpoints for today's Reponsive design:

- Best practice is to start with **mobile first**

Breakpoints

Media queries are used to define breakpoints for today's Reponsive design:

- Best practice is to start with **mobile first**
- Add **min-width** media queries for breakpoints

Breakpoints

Media queries are used to define breakpoints for today's Reponsive design:

- Best practice is to start with **mobile first**
- Add **min-width** media queries for breakpoints
- Add as few as possible and use custom ones

Device-specific breakpoints

... are used to target specific devices, e.g. iPhone 6+, iPhone X or Samsung Galaxy S3.

DEVICE-SPECIFIC BREAKPOINTS

... are used to target specific devices, e.g. iPhone 6+, iPhone X or Samsung Galaxy S3.

Today there are hundreds of different phones of several generations. With different screen sizes, resolutions and ratios. Device-specific breakpoints are just not practical anymore.

ORIENTATION AND RETINA

a more generic use of media queries

```
/* Portrait */
@media screen and (orientation:portrait) { /* Portrait styles here */ }

/* Landscape */
@media screen and (orientation:landscape) { /* Landscape styles here */ }

/* Non-Retina */
@media screen and (-webkit-max-device-pixel-ratio: 1) {
}

/* Retina */
@media only screen and (-webkit-min-device-pixel-ratio: 1.5),
only screen and (-o-min-device-pixel-ratio: 3/2),
only screen and (min--moz-device-pixel-ratio: 1.5),
only screen and (min-device-pixel-ratio: 1.5) {
}
```

Retina

Retina is Apple's brand name for double- and high-density screens. All manufacturers create similar displays.

Retina

Retina is Apple's brand name for double- and high-density screens. All manufacturers create similar displays.

Pixel-based images will look rasterized on high-density screens unless they have **four times the size** they are displayed at.

Best practice

Don't target devices, add breakpoints when the design breaks (custom breakpoints).

Use `em` unit instead of pixels to define a breakpoint.

More on highly effective media queries

Don't forget vertical breakpoints

Vertical breakpoints can come in handy, e.g. using multicol layouts or big typo.

Media queries Level 4

... are currently drafted and the first browsers are implementing parts of it. Check out caniuse.com to see which browsers already support which media query.

```
@media (pointer:coarse) {
  .which-pointer::after {
    content: "Are you on a touchscreen device?";
  }
}

@media (pointer:fine) {
  .which-pointer::after {
    content: "Are you using a mouse or trackpad?";
  }
}
```

Can I hover?

```css
@media (hover) {
  .can-i-hover::after {
    content: "You look like you can hover.";
  }
}

@media (hover:none) {
  .can-i-hover::after {
    content: "I don't think you can hover.";
  }
}
```

Image breakpoints

Next to media queries breakpoints are also used for **responsive images**.

```
<img srcset="elva-fairy-320w.jpg 320w,
             elva-fairy-480w.jpg 480w,
             elva-fairy-800w.jpg 800w"
sizes="(max-width: 320px) 280px,
       (max-width: 480px) 440px,
       800px"
src="elva-fairy-800w.jpg" alt="Elva dressed as a fairy">
```

- MDN Responsive images
- Breakpoints generator

`srcset`

`srcset` defines a list of possible images with its real-size width (with the unit **w**).

`sizes`

`sizes` defines a set of media conditions and which width would be the best to use in these conditions.

What happens then?

Depending on the resolution different sizes of the image will be displayed.
The browser will:

What happens then?

Depending on the resolution different sizes of the image will be displayed.
The browser will:

1. Look at its device width

WHAT HAPPENS THEN?

Depending on the resolution different sizes of the image will be displayed.
The browser will:

1. Look at its device width
2. Work out which media condition in the sizes list is the first one to be true

What happens then?

Depending on the resolution different sizes of the image will be displayed.
The browser will:

1. Look at its device width
2. Work out which media condition in the sizes list is the first one to be true
3. Look at the slot size given to that media query

What happens then?

Depending on the resolution different sizes of the image will be displayed.
The browser will:

1. Look at its device width
2. Work out which media condition in the sizes list is the first one to be true
3. Look at the slot size given to that media query
4. Load the image referenced in the srcset list that most closely matches the chosen slot size

Art direction with a picture

The `picture` tag gives you more control over which image will be displayed.

```
<picture>
  <source media="(max-width: 799px)" srcset="elva-480w-close-portrait.jpg">
  <source media="(min-width: 800px)" srcset="elva-800w.jpg">
  <img src="elva-800w.jpg" alt="Chris standing up holding his daughter Elva">
</picture>
```

Be as relative as possible

- Use realtive units: `em`, `rem` and `vw`, `%`
- Use `clamp`

`clamp`

- `clamp` takes 3 parameters: minimum, preferred and maximum value
- Depending on the viewport width it selects a middle value within the range of minium and maximum value

Good tutorial about `clamp`

`clamp`

```
img {
        width: clamp(400px, 60vw, 600px);
}
```

`clamp`

```
img {
        width: clamp(400px, 60vw, 600px);
}
```

- **400px** is the minimum value.
- **600px** is the maximum value.
- **60vw** is the flexible middle value within the bounds.

clamp

```
* {
        font-size: clamp(1.1rem, 0.7153rem + 1.6368vw, 1.5rem);
}
```

`clamp`

```
* {
        font-size: clamp(1.1rem, 0.7153rem + 1.6368vw, 1.5rem);
}
```

- The `font-size` will be set at `1.1rem`.

`clamp`

```
* {
        font-size: clamp(1.1rem, 0.7153rem + 1.6368vw, 1.5rem);
}
```

- The `font-size` will be set at `1.1rem`.
- Until the computed value of `0.7153rem + 1.6368vw` becomes greater.

`clamp`

```
* {
        font-size: clamp(1.1rem, 0.7153rem + 1.6368vw, 1.5rem);
}
```

- The `font-size` will be set at `1.1rem`.
- Until the computed value of `0.7153rem + 1.6368vw` becomes greater.
- At this point, the `font-size` value will be calculated by the formula of `0.7153rem + 1.6368vw`.

`clamp`

```
* {
        font-size: clamp(1.1rem, 0.7153rem + 1.6368vw, 1.5rem);
}
```

- The `font-size` will be set at `1.1rem`.
- Until the computed value of `0.7153rem + 1.6368vw` becomes greater.
- At this point, the `font-size` value will be calculated by the formula of `0.7153rem + 1.6368vw`.
- Until this preferred value's computed value becomes greater than that of `1.5rem`.

`clamp`

```
* {
        font-size: clamp(1.1rem, 0.7153rem + 1.6368vw, 1.5rem);
}
```

- The `font-size` will be set at `1.1rem`.
- Until the computed value of `0.7153rem + 1.6368vw` becomes greater.
- At this point, the `font-size` value will be calculated by the formula of `0.7153rem + 1.6368vw`.
- Until this preferred value's computed value becomes greater than that of `1.5rem`.
- Now the `font-size` will be set at `1.5rem`.

CSS VARIABLES

```
:root {
        --clr-primary: #e19785;
        --clr-body: #1c1c1c;
        --clr-bg: #ffffff;
        --clr-footer: #ededed;
        --clr-image: #c5c5c5;

        --base-offset: clamp(3rem, -0.9364rem + 10.9344vw, 14rem);
        --base-offset-h: 0 clamp(1.5rem, -0.4682rem + 5.4672vw, 7rem);
        --base-offset-v: 2.5rem 0 2.5rem;
        --base-width: 136.6rem;
}
```

How to start

Tips and tricks

HTML

- Start with a notepad and note the different elements you see, e.g. Navigation, Slider, Teaser, Testimonials
- If elements have similar content and only different styling, they are the same element, e.g. teasers. Try to use the same tags.
- Group the elements together if needed, e.g. lists, articles, sections
- If you need container elements ONLY for styling without any semantic reason, use span and div.

CSS

- Look for similarities and majorities: Ignore the differences for now, look at what is the same on the whole layout, e.g. scrolling text, buttons, headlines
- Once you found all similarities, look for explicit differences
- Go from big to small, and from outside to the inside

Recap: Programming 101

Let's have a look at your solutions

Present Tuesday's exercises:

- List inputs, process and outputs
- Write test cases
- Find constraints
- Write the algorithm in pseudocode

1. Say hello (10 min)

Create a program that prompts for your name and prints a greeting using your name.

2. Counting the number of characters (20 min)

Create a program that prompts for an input string and displays output that shows the input string and the number of characters the string contains.

3. Printing quotes (45 min)

Create a program that prompts for a quote and an author. Display the quotation and author as shown here:
[Author] says, "[Quote]" (Replace [Author] and [Quote] with the actual values.

4. Mad Lib (45 min)

Definition: Mad Libs are a simple game where you create a story template with blanks for words. You or another player then construct a list of words and place them into the story, creating an often silly or funny story as result.

More info on next page

4. Mad Lib (45 min)

Program statement: Create a simple mad-lib program that prompts for a noun, a verb, an adverb, and an adjective and injects those into a story that you create.