[Lesson 2-3]

Roi Yehoshua 2018

[What we learnt last time]

- CSS preprocessors
- SASS
- CSS methodologies
- BEM

[Our targets for today]

- What is Pixel perfect
- Why is it important
- Tools for implementing Pixel perfect
- Adaptive and responsive layout
- Cross-Browser
- Graceful Degradation and Progressive Enhancement

[Pixel Perfect]

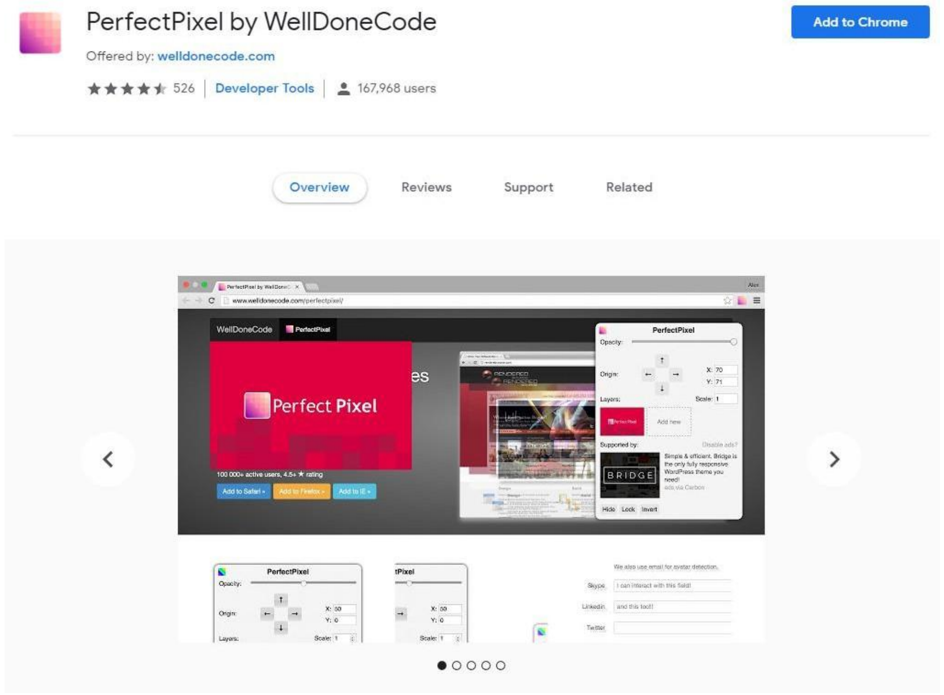
- → Pixel Perfect FrontEnd development means creating a web page as close as the mockup that a UI/UX designer has provided
- → It is the designer’s job to create a Pixel Perfect Design using Photoshop, Illustrator, Sketch or other tool
- → Front-end developer needs to convert a pixel-perfect mockup to code

[PerfectPixel plugin]

- ➞ There are several useful tools for pixel perfect development
- ➞ PerfectPixel plugin for Google Chrome is a powerful tool that allows seeing difference between mockup from the designer and the website that you create
- ➞ It allows developers to put a semi-transparent image overlay over the top of the developed HTML, and perform pixel perfect comparison between them

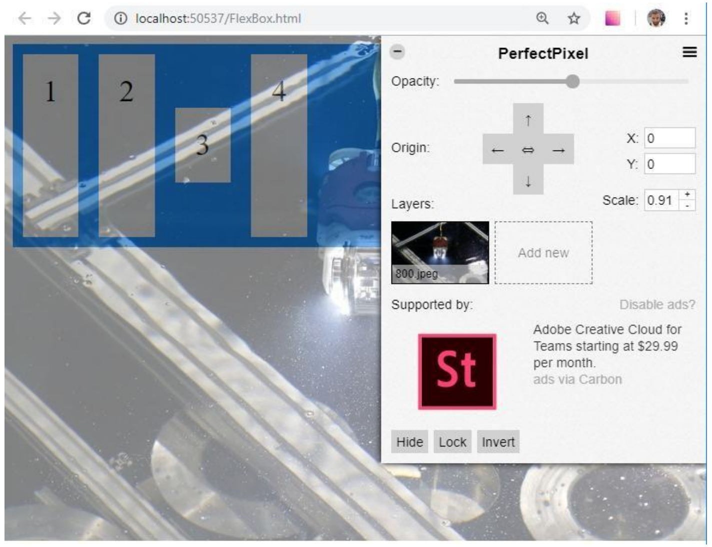
[PerfectPixel plugin]

→ Install the plugin from google chrome store


PerfectPixel by WellDoneCode
Offered by: welldonecode.com
526 | Developer Tools | 167,968 users
Add to Chrome
Overview
Reviews
Support
Related

[PerfectPixel plugin]

- → After installing the plugin, you'll see the "Perfect Pixel Icon" on the right navbar
- → When you click it, the drop zone of an image will show up
- → You just have to drag&drop your mockup there
- → Then the mockup will appear above your website, so you can compare them
- → The opacity and position of the top layer are adjustable



[Pixel perfect development]

- Creating Pixel perfect web page is usually done in several stages:
  - Writing HTML/CSS code to make page look similar to a mockup
  - Comparing the result with a mockup using tools like PerfectPixel plugin
  - Adjusting CSS code for a top left element of the page (usually contains company logo) to be perfectly aligned with the mockup. Later this element will be used as an anchor when adjusting position of the overlay image
  - Adjusting CSS code for other elements on the page, from top to bottom, and from left to right inside the container

[Modern Web]

- ➔ In the past decade variation of target resolutions and devices have significantly increased
- ➔ Because of this sites can't be built targeting only common desktop resolutions
- ➔ Not targeting a wide range of devices may result in a significant fall of site experience quality and it's popularity
- ➔ Two major approaches have been developed to address this challenge - responsive and adaptive

Responsive layout

- ➔ The responsive approaches don't target any specific device or resolution
- ➔ Instead, it's philosophy is built on site responding to any possible environment
- ➔ The responsive site is a flexible site based on relative sizes
- ➔ Site built in a responsive way is future-proof, as it intended to look good on any environment including not released yet devices with unknown resolution or other params

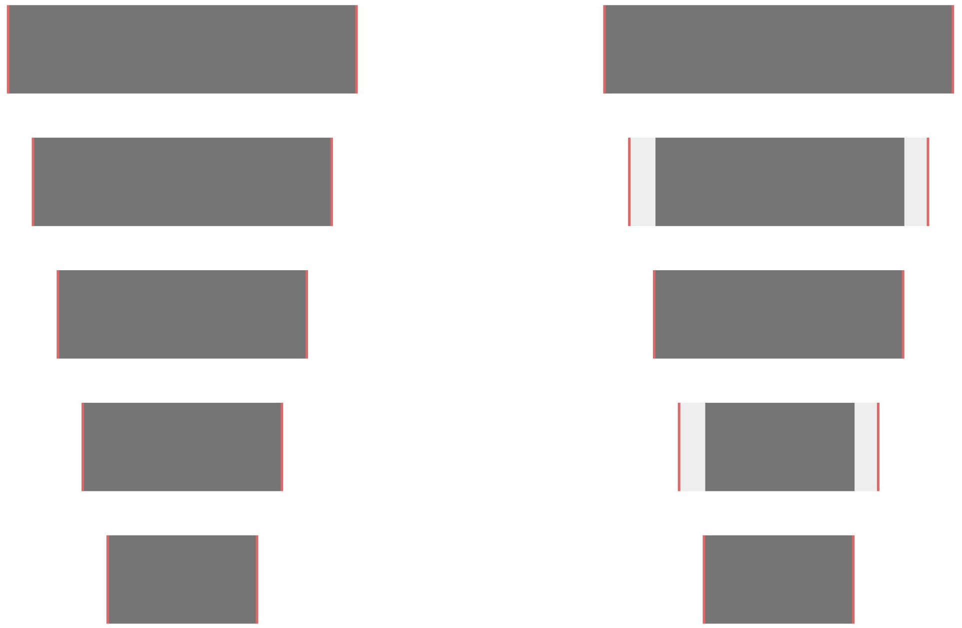
Adaptive layout

- ➔ The adaptive approach is built on choosing target common devices and resolutions and building support for them first
- ➔ The Adaptive approach may result in a better look on target devices than responsive
- ➔ On the other hand, adaptive site may look worse on non-target devices
- ➔ Choosing the adaptive approach requires continuous support as future common devices could have different resolutions or other params

[Content handling in responsive and adaptive]

Responsive approach

Adaptive approach



← Common large resolution

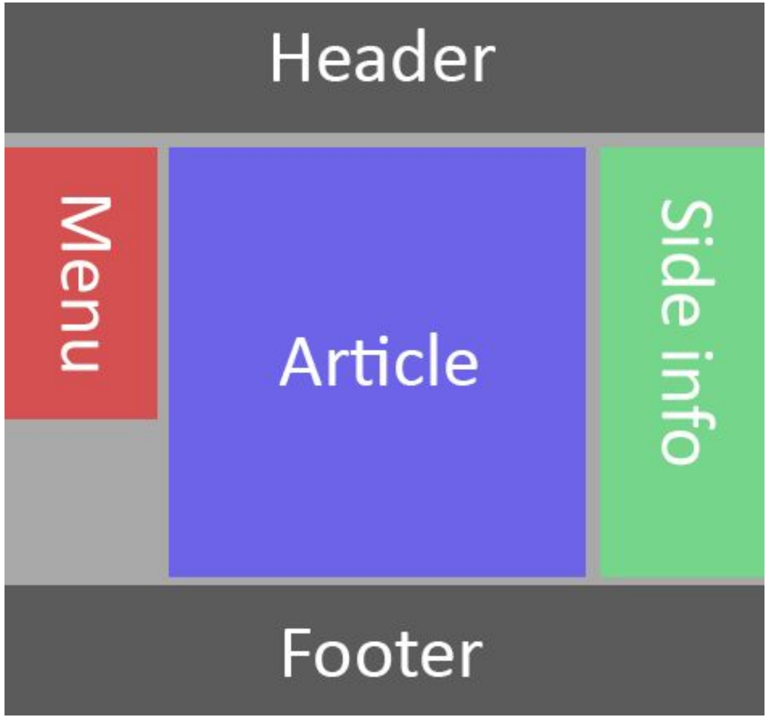
← Rare in-between resolution

← Common medium resolution

← Rare in-between resolution

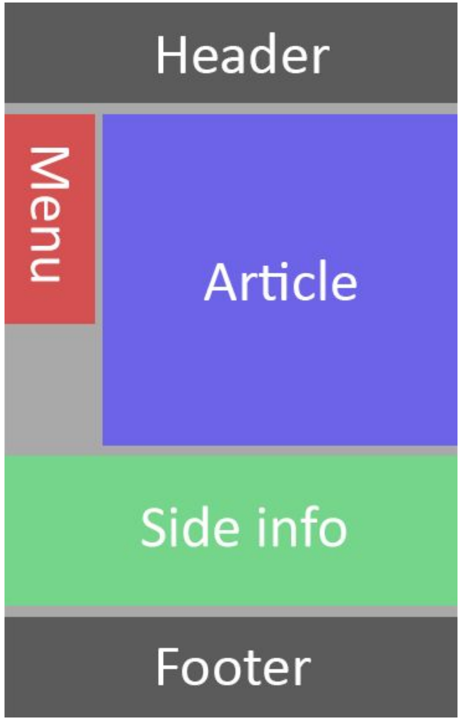
← Common small resolution

Device layout - desktop



- ➔ This is one of the possible layouts for the desktop web page
- ➔ On the desktop, we have sufficient space, so it’s possible to place menu, article and some side info in one row
- ➔ The menu may or not be fixed
- ➔ We may use large images or video
- ➔ Take care that font size is not too small

[Device layout - tablet]

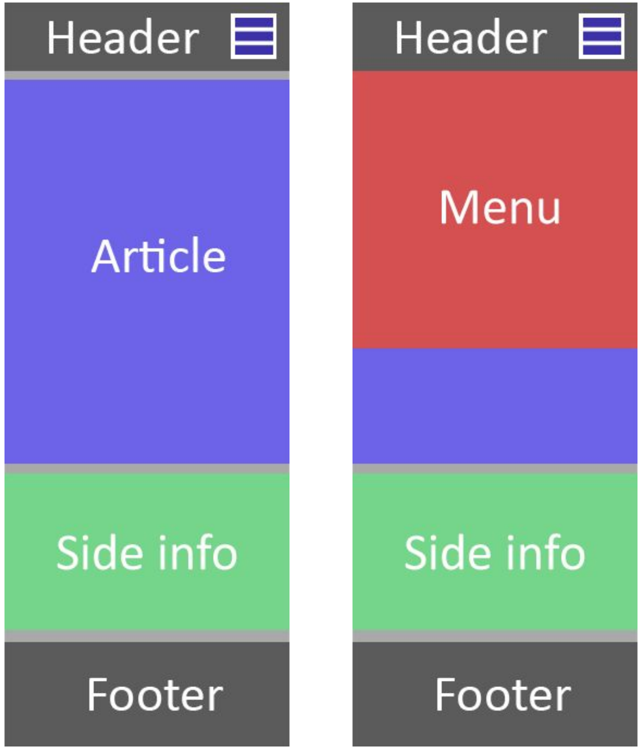


➔ On the tablet, we free space for the article section by moving side info below it

➔ Make sure the font size is not small nor big

➔ Use smaller size images

[Device layout - mobile]



- ➔ On mobile, we hide the menu by default
- ➔ Instead, we add a menu icon. Touching it toggles menu visibility
- ➔ Header with menu icon is fixed in the top of the screen and is accessible on any scrolling position
- ➔ Any media must be optimized for small screens and mobile internet connection
- ➔ Make sure the font is readable

[Viewport meta]

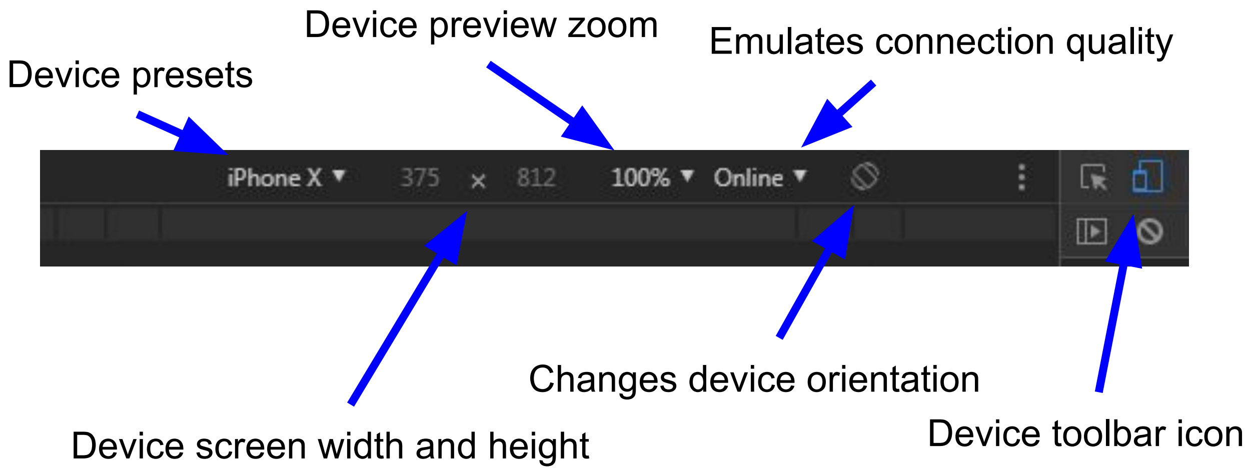
- ➔ Mobile devices adapt to desktop sites by rendering it in a bigger resolution that they have and then shrinking it down to fit it in the viewport
- ➔ If not instructed specifically by meta-tags, a mobile device will assume the site is designed for the desktop. So media queries for smaller screens may not work
- ➔ Meta information "viewport" allows a site to tell the mobile device how it should render and scale it

  - ➙ Example of common viewport-meta info to declare that the site is designed for mobile and no scaling is required:

```
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

[Previewing and Testing]

- ➔ Google Chrome provides tool for device-specific testing
- ➔ Open development console and click on the device toolbar icon - this will open device testing mode


Device preview zoom
Emulates connection quality
Device presets
iPhone X ▼
375
×
812
100% ▼
Online ▼
Changes device orientation
Device toolbar icon
Device screen width and height

Previewing and Testing

- ➔ Testing tools do not guarantee that the site will look the same on target device
- ➔ There are many factors for that like:
  - → Different browsers or browser versions
  - → Different fonts or font smoothing
  - → Predefined mobile OS input element styles
  - → Difference between emulation of touch and actual device touch events

- ➔ Because of this all final tests must be done on actual target devices to assume good quality of the site experience.

Media Queries

- ➔ Specific CSS styles can be applied only for specific target devices
- ➔ This is done using Media Queries
- ➔ Media Queries allows specifying conditions like:
  - ➞ Screen width and height
  - ➞ Screen resolution
  - ➞ Device orientation and aspect-ratio

- ➔ There are much more, but this is the most often used ones

[Adding Media Queries]

➔ In the CSS file and <style> tag

```
@media (max-width: 375px) {
  /* mobile styles */
}
```

➔ Using CSS Import

```
@import url("mobile.css") (max-width: 375px);
```

➔ Linking CSS file in the HTML document

```
<link media="(max-width: 375px)" href="mobile.css" rel="stylesheet">
```

[DAN.IT]
EDUCATION

[Examples of Media Queries]

➔ Changing top menu items order from row to the column on devices with width less than 321px:

```
.top-menu { flex-direction: row; }

@media (max-width: 320px) {
 .top-menu { flex-direction: column; }
}
```

➔ Changing font-size for devices with width from 801px to 600px:

```
.top-menu { font-size: 1.2em; }

@media (max-width: 800px) and (min-width: 600px) {
  .top-menu { font-size: 1em; }
}
```

[Using flexbox]

- ➔ CSS flexbox is very useful for responsive design
- ➔ It can shrink or expand content depending on the free space
- ➔ Unlike block elements, flex content is less likely to get out of its container, making it a very safe tool for creating responsive layouts
- ➔ Some of CSS Flexbox styles that are important for the purpose of responsive layouts:
  - → flex-wrap - will determine if the content will try to shrink if there is not enough space for it, or will it move content to the next row/column
  - → flex-grow - controls if the element can grow if there is extra free space available for it
  - → flex-shrink - controls whether element should shrink if there is not enough space for it
  - → order - sets the order of the element in a row/column. This is very useful since sometimes the order of elements of the component could change in the desktop and mobile designs

[Using CSS grid]

- ➔ CSS Grid proposes a native grid layout
- ➔ Both grid structure and assignment of components to grids can be changed depending on the media queries
- ➔ This provides a very powerful tool for changing the layout of the page for different devices

- ➔ Flexbox or grid are not necessary to create a responsive/adaptive layout
- ➔ But in combination with media-queries flexbox and/or grid can create a very stable and safe layout that will look good almost on any device

[Mobile First]

- ➔ Supporting mobile device requires more than adopting layout for small screens
- ➔ Other important aspects of supporting the growing mobile market are covered by the Mobile First concept

- ➔ Its important aspects are:
  - → The same site must serve all devices
  - → Page layout must be created for mobile devices first, and then adapted for the higher resolutions
  - → The user interface must be comfortable on every device
  - → A user must first get only the most important information and get more on his request
  - → A site must be lightweight. Optimize site load time
  - → Make a smaller version of the big images specifically for mobile

[Touchscreen]

- ➔ Touchscreen and mouse, while similar, have significant differences
- ➔ CSS hover and mouse over events won't work as smooth as with mouse
- ➔ Always make togglable mobile menu accessible by touching/clicking a specific button, and not by using CSS Hover
- ➔ Instead of **MouseEvent,** javascript has **TouchEvent**, **Touch** and **TouchList** interfaces specifically for touchscreens:
  - → **TouchEvent** - has some similarities to the **MouseEvent** interface, but adapted for touchscreens. It manages events like touch start, touch end, touch move, etc.
  - → **Touch** - represents a single contact point on a touchscreen. It contains data like coordinates and pressure used
  - → **TouchList** - touchscreen may contain more than a single **TouchEvent**. Example - actions with two fingers to zoom view. This interface contains a list of all touch events

[CSS Vendor Prefixes]

- CSS vendor prefixes, aka CSS browser prefixes, are a way for browser makers to support new CSS features before those features are fully supported in all browsers
- This may be done during a testing and experimentation period where the browser manufacturer is determining exactly how to implement these features
- The major browsers use the following prefixes:

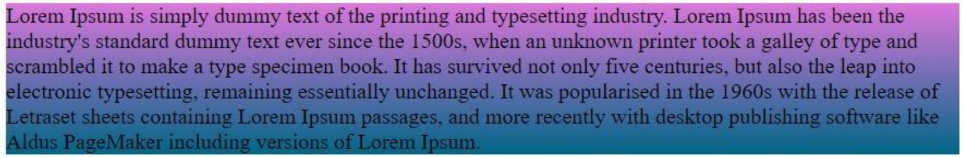
Prefix	**Browsers**
-webkit-	Chrome, Safari, newer versions of Opera, almost all iOS browsers (including Firefox for iOS)
-moz-	Firefox
-ms-	Internet Explorer and Microsoft Edge
-o-	Old, pre-WebKit, versions of Opera

[DAN.IT] EDUCATION

[Vendor Prefixes Example]

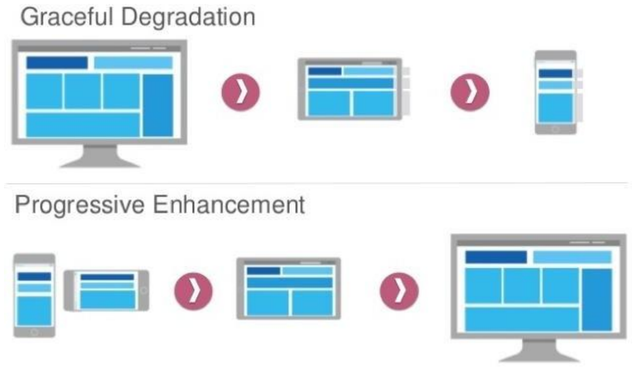
```
<style>
    /* Cross-browser css3 linear-gradient */
    .linear-gradient {
        /* Webkit (Safari, Chrome, iOS, Android) */
        background-image: -webkit-linear-gradient(top, #D7D 0%, #068 100%);
        /* Firefox */
        background-image: -moz-linear-gradient(top, #D7D 0%, #068 100%);
        /* Opera */
        background-image: -o-linear-gradient(top, #D7D 0%, #068 100%);
        /* Older Webkit syntax */
        background-image: -webkit-gradient(linear, left top, left bottom, color-stop(0, #D7D),
color-stop(1, #068));
        /* W3C */
        background-image: linear-gradient(to bottom, #D7D 0%, #068 100%);
    }
                                                 </style>
```

```
<div class="linear-gradient">  Lorem
    Ipsum …
</div>
```


Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

[Graceful Degradation vs. Progressive Enhancement]

- **Graceful degradation** is the practice of building web sites that provides a certain level of user experience in more modern browsers, but degrades gracefully to a lower level of user experience in older browsers (things don't break for them)
- **Progressive enhancement** means that you start by establishing a basic level of user experience that all browsers will be able to provide, but also build a more advanced functionality that will automatically be available to browsers that can use it



[Graceful Degradation vs. Progressive Enhancement]

- For example, let’s say we would like to provide our users a “print this page” link
- Although browsers provide this option via their menus, user testing shows that as a last step in a booking process, such links are a good re-affirming call to action
- Implementing this functionality in JavaScript is easy - the window object of the browser has a print() method that can be called to start a printout
- Probably the most common way of doing that is by using the javascript: protocol:

```
<p id="print">
    <a href="javascript:window.print()">Print this page</a>
</p>
```

- This works when JavaScript is available and enabled, and the browser supports the print command
- However, if JavaScript is not available (for example on some mobile devices) then this link will not work. This delivers a bad user experience.

[DAN.IT] EDUCATION

[Graceful Degradation vs. Progressive Enhancement]

- In the **graceful degradation** approach, we can tell the user that the link may not be working and what the reason for that is, and even suggest an alternative solution to achieve what they want to do.
- A common trick is to use the **<noscript>** element
  - Anything inside this element will be shown when JavaScript is not available

```
<p id="print">
    <a href="javascript:window.print()">Print this page</a>
</p>
<noscript>
    <p class="scriptwarning">
        Printing the page requires JavaScript to be enabled.
        Please turn it on in your browser.
    </p>
</noscript>
```

- This is considered degrading gracefully - we explain to the user that something is wrong and how to work around the issue

[DAN.IT] EDUCATION

[Graceful Degradation vs. Progressive Enhancement]

- However, this assumes that visitors to your site:
  - Know what JavaScript is
  - Know how to enable it
  - Have the rights and the option to enable it
  - Feel happy about turning on JavaScript just to print a document
- A slightly better approach would be:

```
<p id="print">
    <a href="javascript:window.print()">Print this page</a>
</p>
<noscript>
    <p class="scriptwarning">
        Print a copy of your confirmation.
        Select the "Print" icon in your browser, or select "Print" from the "File" menu.
    </p>
</noscript>
```

[DAN.IT]
EDUCATION

[Graceful Degradation vs. Progressive Enhancement]

- Using the **progressive enhancement** approach, the first step would be to find out if there is a non-scripting way of performing the operation. In our case there isn't.
- The second step is to not assume that the user has JavaScript enabled and that the browser can print
- Instead we just tell the user t that they need to print the document and leave the "how" up to them:

```
<p id="print">
    Thank you for your order. Please print this page for your records.
</p>
```

- Then we use **unobtrusive JavaScript** to add a print button only when the browser supports it:

Graceful Degradation vs. Progressive Enhancement

```
<script>
                           (function () {
        if (document.getElementById) {
            let pt = document.getElementById('print');
            if (pt && typeof window.print === 'function') {  let btn
                = document.createElement('button');
                btn.setAttribute('type', 'button');  btn.innerHTML =
                'Print this now';  btn.onclick = function () {
                    window.print();
                }
                pt.appendChild(btn);
            }
        }
        })();
</script>
```

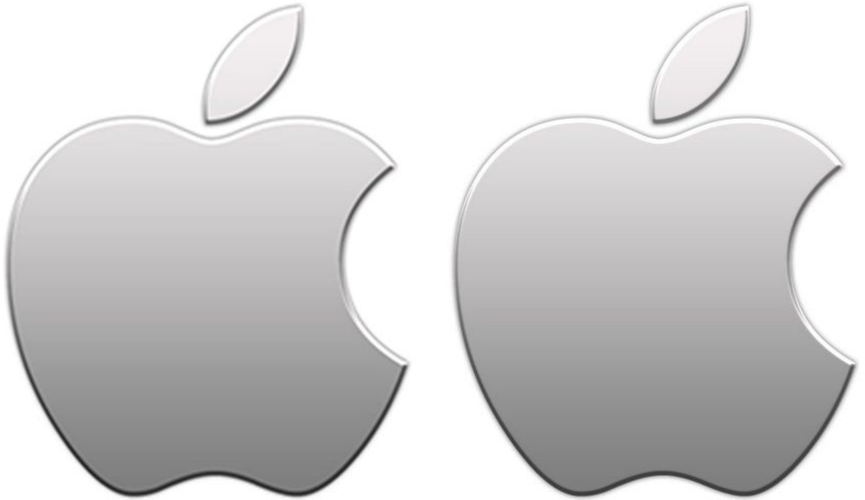
- → By wrapping the whole functionality in an anonymous function and immediately executing it, we don't leave any global variables behind
- → We test for DOM support and try to get the element we want to add the button to
- → We then test if the element exists and if the browser has a window object and a print method
- → If both are true, we create a new click button and apply window.print() as the click event handler

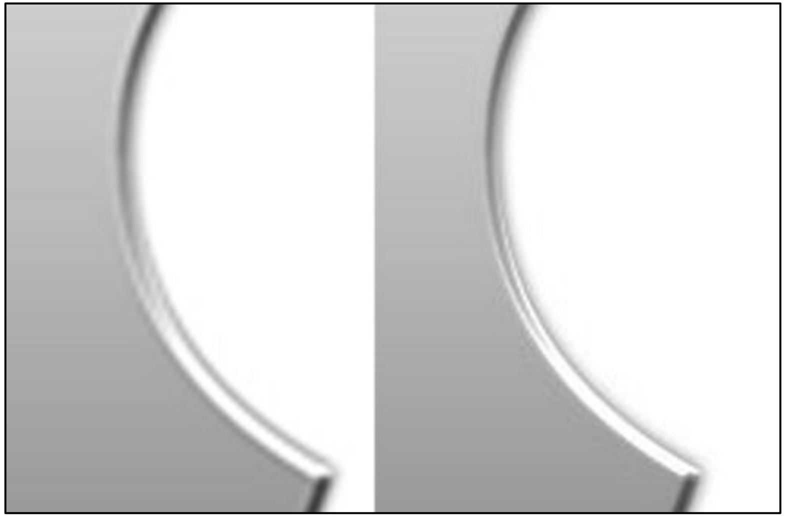
[Retina]

- ➔ Retina technology was introduced by Apple for their mobile devices and since then many other manufacturers started to use similar technology.
- ➔ The retina is a double-density screen. When one pixel actually consists of a group of pixels.
- ➔ So a screen with a 2880x1800 resolution will work as a 1440x900 screen.
- ➔ Extra pixels provides better smoothness, better font display and making image overall more pleasant to eye on compact screens.

[Retina]

- ➔ To feel the difference between the scaled size and actual size look at two images of the same scale.
- ➔ The right one has twice as many pixels. Notice higher details and better smoothness.





[Terminology]

- ➔ Retina - Apple brand name for their own high-density screen devices.
- ➔ High density screen/device - common names for similar non-Apple solutions.

- ➔ Device/Hardware/Physical pixels - the real pixels of the device. The most atomic unit of the screen device.
- ➔ CSS/Device independent pixels - pixel size of the content. Could differ when measured in real device pixels.

[Terminology]

- ➔ Pixels per inch - the number of real pixels per inch of the screen device.
- ➔ Dots per inch - PPI is a more accurate term for screen devices, while DPI comes from the polygraphy. Still, it is being very often used as a synonym of PPI.

- ➔ Device/CSS pixel ratio - a common term for the ratio of the size of one physical pixel to the size of one CSS pixel.
- ➔ Dots per pixel (dppx) - is the same as a device pixel ratio. This term is used by the W3C in their specifications.

[Examples]

Apple iPad 4

Diagonal	9.7"
Resolution	2048×1536
DPI	264
dppx	2

Amazon Kindle Fire HD 8.9

Diagonal	8.9"
Resolution	1920×1200
DPI	254
dppx	1.5

Samsung Galaxy S6

Diagonal	5.1"
Resolution	1440×2560
DPI	576
dppx	4

[High resolution images]

- ➔ While small images could look good on small resolutions, on a similar HD screen they will look blocky.
- ➔ To counter this a higher-resolution version of images often being used.
- ➔ Different versions of the same images for a different pixel-ratios could be used.
- ➔ Media query resolution with a DPI unit allow to set CSS rules for a specific screens depending on their DPI value.
- ➔ Another way is to use device-pixel-ratio media query - it allows setting CSS rules for high density displays depending on their dppx.
  - → Note, that device-pixel-ratio is a non-standard query at this moment.

[High resolution images]

➔ You could work with the high-resolution images and use automation tools like webpack/gulp/grunt to automate the creation of downscaled images for the small and non-retina resolutions.

➔ @2 is an Apple proposed and often used suffix for the double density versions of images.

```
@media screen and (min-resolution: 192dpi) {
  img.promo {
     content:url("http://example.com/promo@2x.jpg");
  }
}

@media screen and (min-device-pixel-ratio: 1.5) {
  #heroimage {
     background-image: url(sell@2x.jpg);
  }
}
```

[Retina and JavaScript]

- ➔ window.devicePixelRatio is a read-only value that contains dppx and can be used to write scripts specific for high-density screens.
- ➔ It is also possible to use CSS media queries in Javascript via window.matchMedia - it receives as argument a CSS media query in a form of a string and returns MediaQueryList object.
  - → MediaQueryList object contains value matches, that is set to true if media query is valid.

```
if( window.devicePixelRatio > 1 ) {
   //HD Screen script
}

if( window.matchMedia('only screen and (min-resolution: 192dpi)').matches ) {
   //HD Screen script
}
```

[Fonts and SVG images]

- ➔ SVG fonts and icons being a vector-based make them DPI independent.
- ➔ They describe shapes and not a pixel placement. So they can be scaled to any size without quality loss.
- ➔ If you need to support a wide variety of screen resolutions and device pixel ratios - SVG fonts and icons is a great solution.

- ➔ Another great tool for supporting high-density screen is a CSS graphics.
- ➔ You could use raster images for things like gradients, shadows, and other effects. But doing the same with CSS will make your design adaptive to all kind of screen devices.

[DAN.IT] EDUCATION

Links

- ➔ SVG fonts and icons online databases
  https://fontawesome.com
  https://www.s-ings.com/typicons/
  https://zurb.com/playground/foundation-icons

- ➔ Tools and libraries
  http://imulus.github.io/retinajs/ - a library for working with double dencity images
  http://dpi.lv - catalog of mobile devices with their resolution, DPI and dppx values

[Control questions]

1. Explain what is Pixel perfect development
2. Why is Pixel perfect important and when it could be used?
3. What tools can we use to make Pixel perfect page?
4. Why is it important to adjust CSS code for blocks from top to bottom and from left to right during Pixel perfect development
5. What is the difference between adaptive and responsive approaches?
6. Why is such approaches are needed?
7. How can we change page layout on mobile version?
8. What is viewport meta for?
9. How can we test site for various target devices?
10. What is media queries and how can we add them?

[Control questions]

11. Name some Vendor Prefixes and explain why are they being used.
12. What is Graceful Degradation?
13. What is Progressive Enhancement?