



UNIVERSITÀ DI PAVIA

Collegio Alessandro Volta  
Via Adolfo Ferrata, 17, Pavia (PV)



# WEB DESIGN

Lecture 4 – Responsive Design & Websites

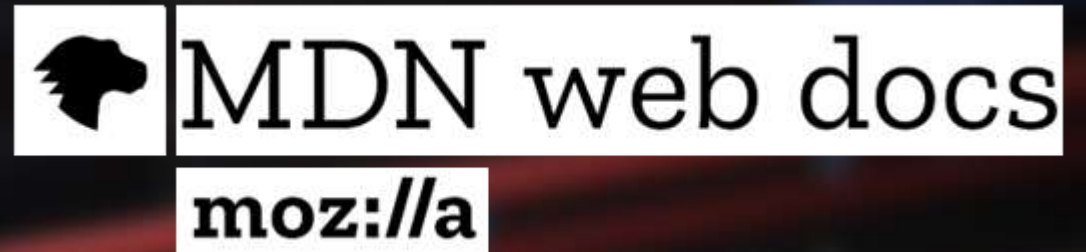
**Giovanni Nicola D'Aloisio**

Department of Physics – University of Pavia  
Classe di Scienze, Tecnologie e Società – IUSS Pavia

E-Mail: [giovanninicola.daloisio01@universitadipavia.it](mailto:giovanninicola.daloisio01@universitadipavia.it)

# Try it by yourself

- Typesetting a community school homepage;
- Styling a biography page;
- Fundamental CSS comprehension;
- Creating fancy letterheaded paper;
- A cool-looking box;
- Fundamental layout comprehension.



Most important

You'll build this website, then you'll apply it the things we'll have seen at the end of this lecture.

# What accessibility means

When someone describes a site as "accessible", they mean that any user can use all its features and content, regardless of how the user accesses the web — especially users with impairments.

Sites should be accessible to keyboard, mouse, and touch screen users, and any other way users access the web, including screen readers and voice assistants like Alexa, Siri and Google Home.

Applications should be understandable and usable by people regardless of auditory, visual, physical, or cognitive abilities. Sites should also not cause harm: web features like motion can cause migraines or epileptic seizures.





# HTML & Accessibility

As you practice more your HTML skills you'll keep seeing a common theme: the importance of using semantic HTML. You have to use the correct HTML tags for their intended purpose as much as possible.

You might wonder why this is so important. After all, you can use a combination of CSS and JavaScript to make just about any HTML element behave in whatever way you want, like a control button to play a video on your site (you might just use a CSS/JavaScript-stylized `<div>` element).

Try to use this code, and then try to move the web page. You should see the different behavior of the elements.

## HTML

```
<div class="button">Something</div>
```

```
<button class="button button1">Green</button>  
<button class="button button2">Blue</button>
```

## CSS

```
.button {border: none; color: black; padding: 16px  
32px; text-align: center; text-decoration: none;  
display: inline-block; font-size: 16px; margin: 4px  
2px; transition-duration: 0.4s; cursor: pointer;}
```

```
.button1 {background-color: white; color: black;  
border: 2px solid #4CAF50;}
```

```
.button1:hover {background-color: #4CAF50;  
color: white;}
```

```
.button2 {background-color: white; color: black;  
border: 2px solid #008CBA;}
```

```
.button2:hover {background-color: #008CBA; color:  
white;}
```

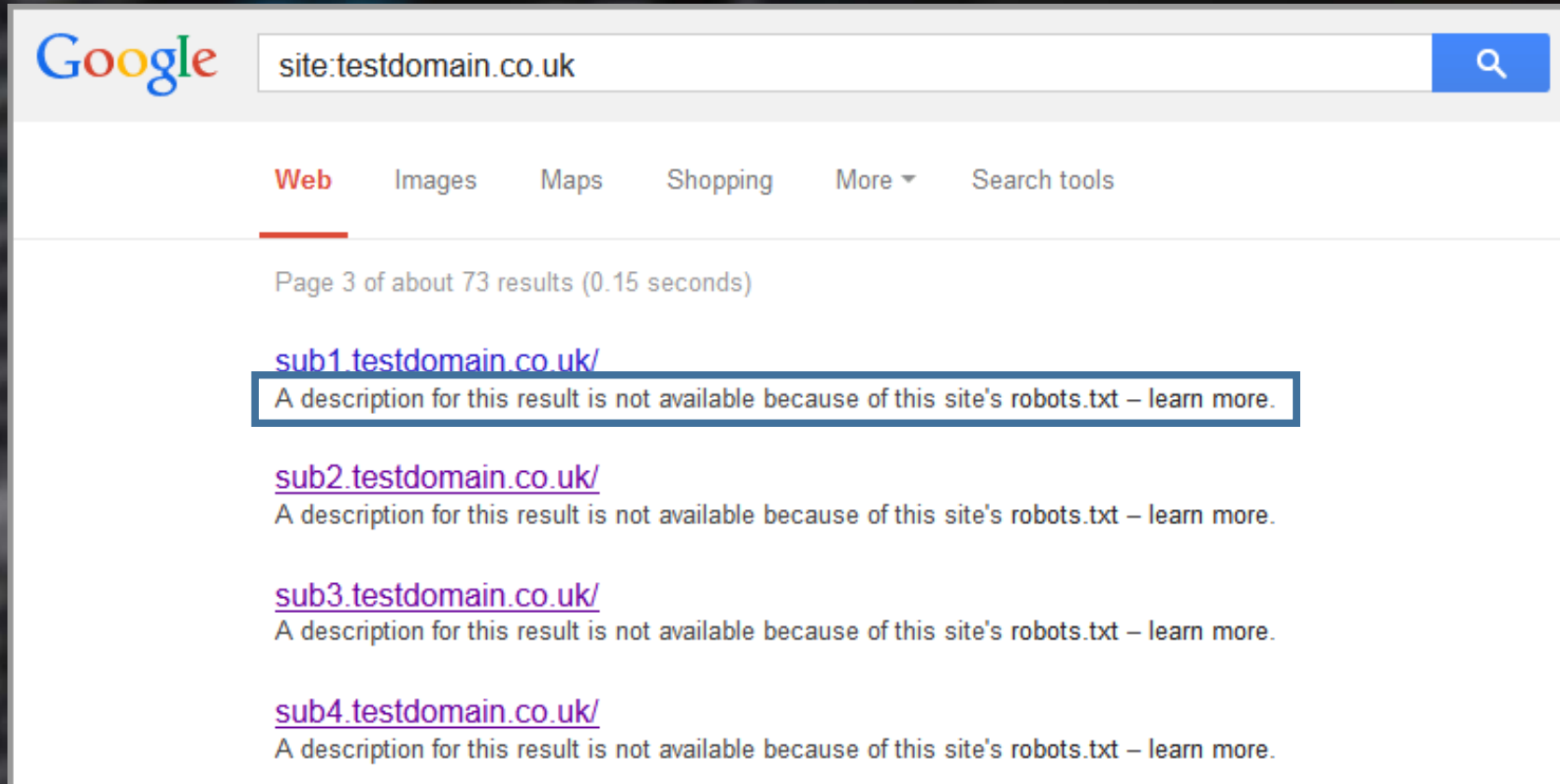
# Semantic HTML

Semantic markup has other benefits beyond accessibility:

- Easier to develop with — you get some functionality for free, plus it is arguably easier to understand.
- Better on mobile — semantic HTML is arguably lighter in file size than non-semantic spaghetti code.
- Good for SEO — search engines give more importance to keywords inside headings, links, etc., so your documents will be more findable by customers.



# robots.txt



More info on [GoogleDev](#)



# The importance of being clear

- One of the best accessibility aids a screen reader user can have is an excellent content structure with headings, paragraphs, lists, etc, like the reported one; anyway, people sometimes write headings, paragraphs, etc. using line breaks and adding HTML elements purely for styling, and you'll not have a very good experience.
- Also, always use clear language that is not overly complex and doesn't use unnecessary jargon or slang terms.
- Use `<abbr>` and similar tags.

## HTML

```
<h1>My heading</h1>
```

```
<p>This is the first section of my document.</p>
```

```
<p>I'll add another paragraph here too.</p>
```

```
<ol>  
  <li>Here is</li>  
  <li>a list for</li>  
  <li>you to read</li>  
</ol>
```

```
<h2>My subheading</h2>
```

```
<p>  
  This is the first subsection of my document.  
</p>
```

```
<h2>My 2nd subheading</h2>
```

```
<p>  
  This is the second subsection of my content.  
</p>
```

# The HTML <meta> element

- The <meta> element is typically used to define the character set, page description, keywords, author of the document, and viewport settings.
- The metadata will not be displayed on the page, but is used by browsers (how to display content or reload page), by search engines (keywords), and other web services.

Character set <meta charset="UTF-8"> [HTML](#)

Keywords <meta name="keywords" content="HTML, CSS, JavaScript"> [HTML](#)

Description <meta name="description" content="Free Web tutorials"> [HTML](#)

Author <meta name="author" content="John Doe"> [HTML](#)

Refresh <meta http-equiv="refresh" content="30"> [HTML](#)

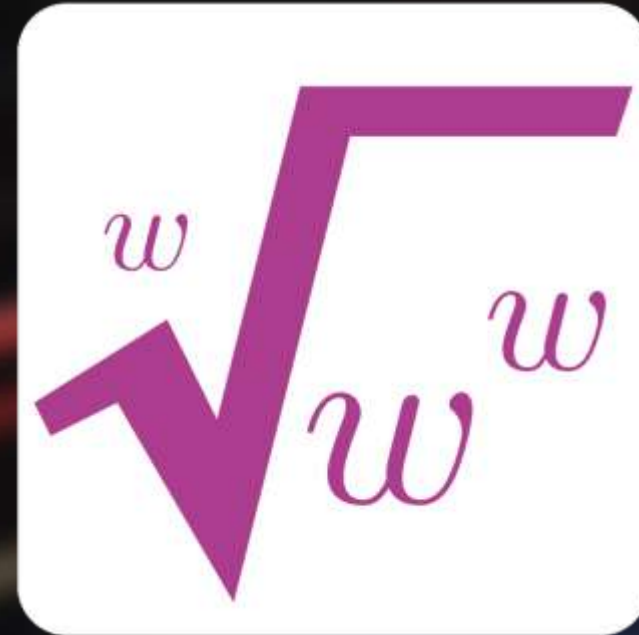
Viewport <meta name="viewport" content="width=device-width, initial-scale=1.0"> [HTML](#)





# Writing mathematics with MathML

Mathematical Markup Language — or MathML — is the markup language used to write mathematical formulas in web pages using. Although it was originally designed as an independent XML language, MathML is generally embedded inside HTML documents and can be seen as an extension of HTML.

More details [here](#).



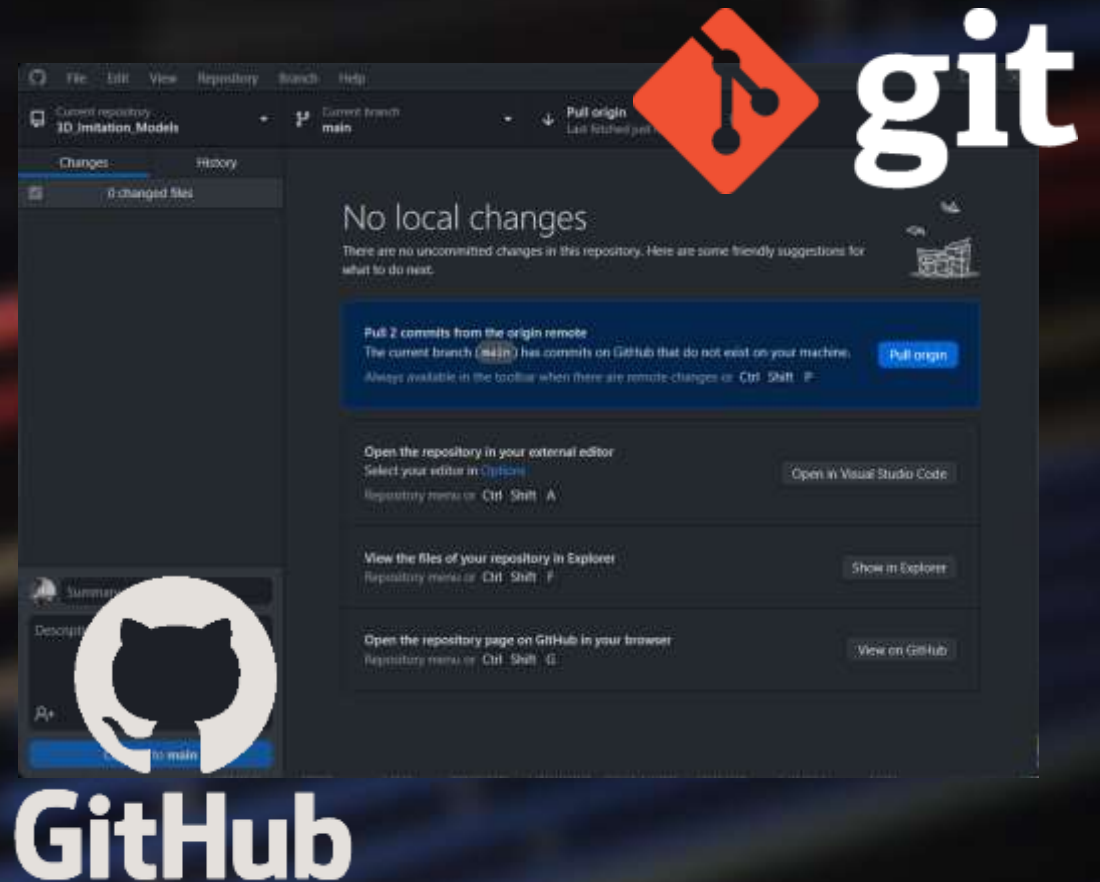
# Cross browser testing

Safari	Chrome
 <p>GROCERIES</p> <p>Email: <input type="text" value="user@nativescript.org"/></p> <p>Passw <div>Your password must contain at least one number.</div> <input type="password" value="....."/></p> <p><input type="button" value="Login"/> <input type="button" value="Sign Up"/></p> <p>Browser: Safari</p>	 <p>GROCERIES</p> <p>Email: <input type="text" value="user@nativescript.org"/></p> <p>Password: <input type="password" value="....."/></p> <p><div><div>! Your password must contain at least one number.</div><input type="button" value="Login"/> <input type="button" value="Sign Up"/></div></p> <p>Browser: Chrome</p>

# Git (& GitHub)

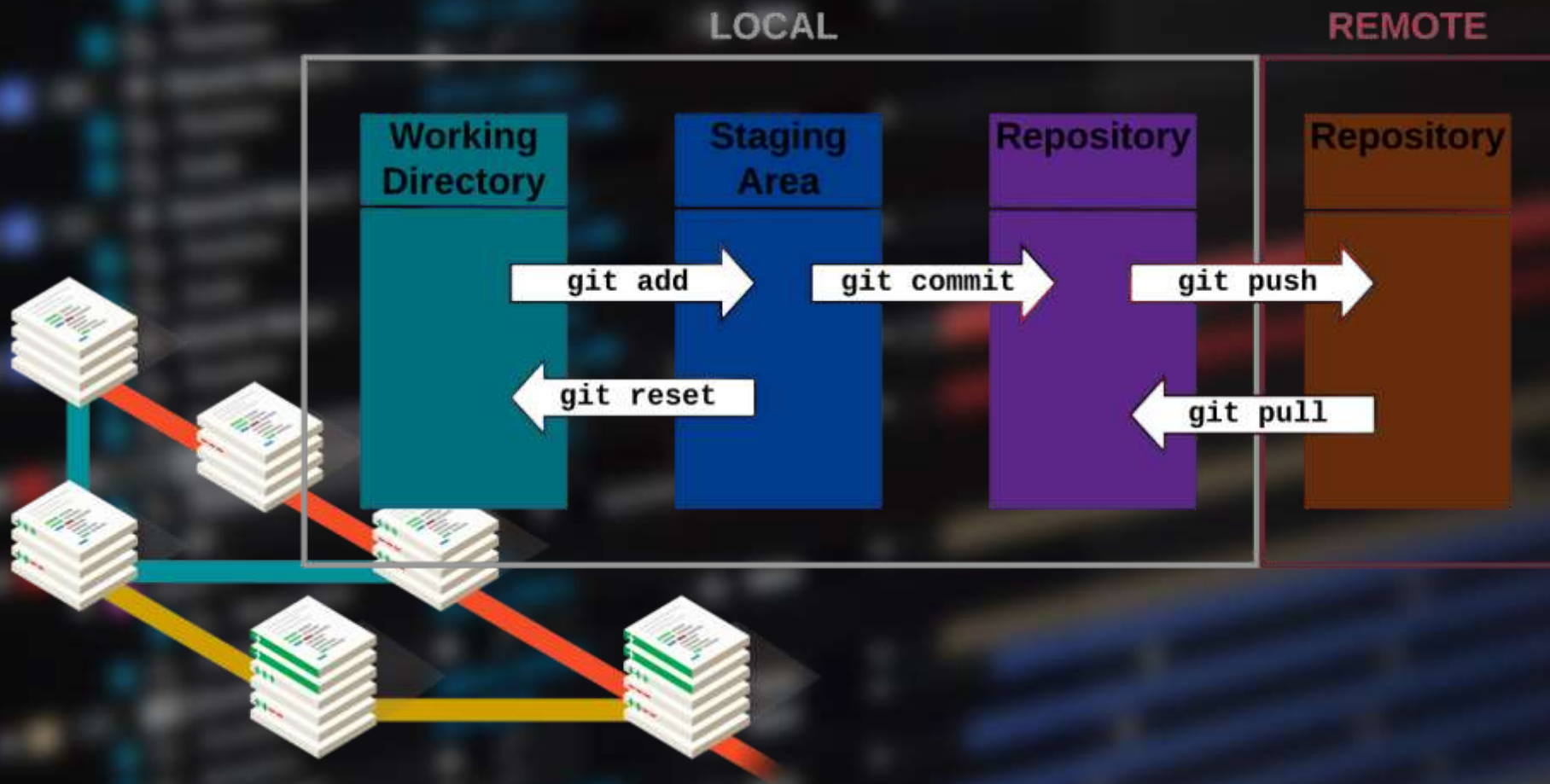
- Version Control System (VCS) – Tool that allows to collaborate with other developers on a project without the danger of them overwriting each other's work, and roll back to previous versions of the code base if a problem is discovered later on.
- Git – Most famous VCS.
- GitHub – A site that provides hosting for your repositories and several tools for working with them.

If you haven't subscribed to [GitHub](#) yet, and if you didn't install [GitHub Desktop](#).





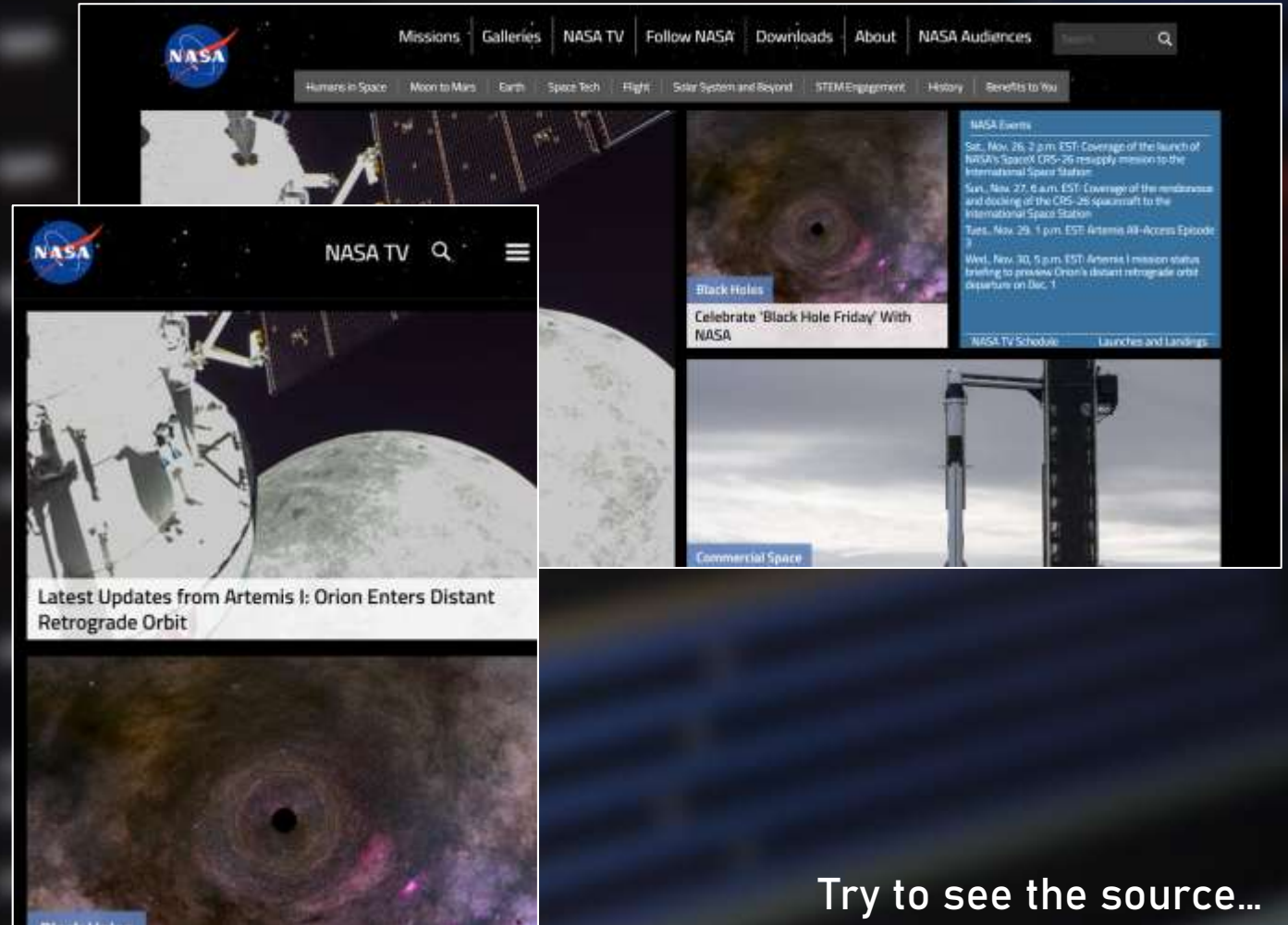
# How Git works



# Responsive web design

The use of mobile devices to surf the web continues to grow at an astronomical pace, and these devices are often constrained by display size and require a different approach to how content is laid out on the screen.

Responsive web design responds to the needs of the users and the devices they're using. The layout changes based on the size and capabilities of the device. For example, on a phone users would see content shown in a single column view; a tablet might show the same content in two columns.



Try to see the source...

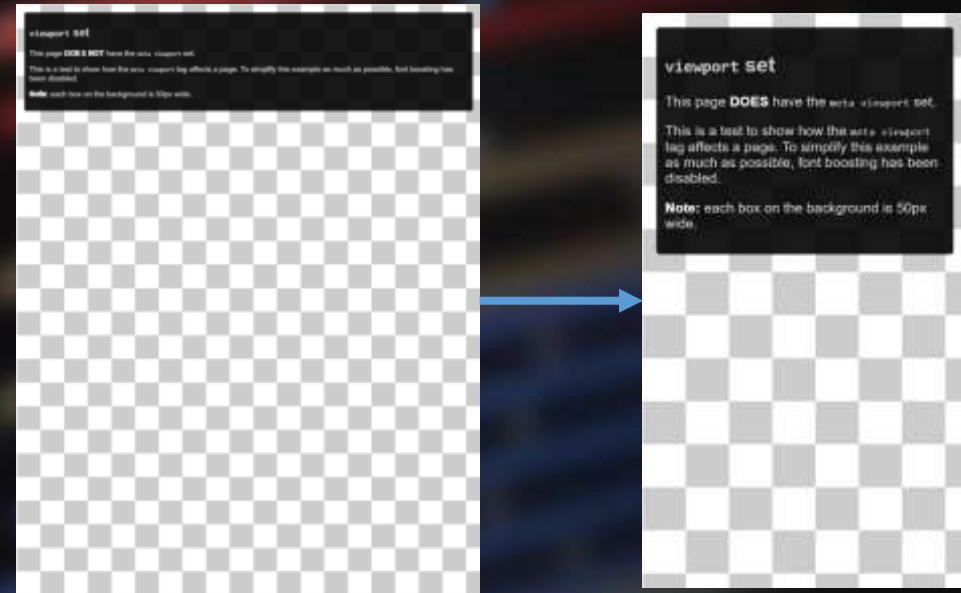
# Set the viewport

A meta viewport tag gives the browser instructions on how to control the page's dimensions and scaling.

Using the meta viewport value `width=device-width` instructs the page to match the screen's width in device-independent pixels. A device (or density) independent pixel being a representation of a single pixel, which may on a high density screen consist of many physical pixels. This allows the page to reflow content to match different screen sizes, whether rendered on a small mobile phone or a large desktop monitor.

## HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    ...
  </head>
  ...
```





# Set the viewport

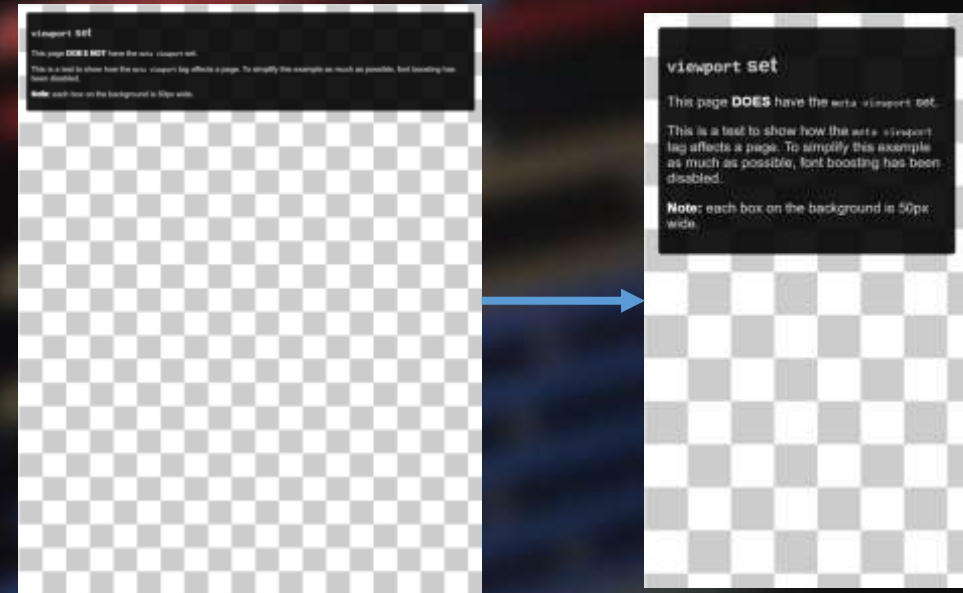
Adding the value `initial-scale=1` instructs browsers to establish a 1:1 relationship between CSS pixels and device-independent pixels regardless of device orientation, and allows the page to take advantage of the full landscape width.

In addition to setting an `initial-scale`, you can also set a `minimum-scale` and a `maximum-scale`, or even an `user-scalable`.

When set, these can disable the user's ability to zoom the viewport, potentially causing accessibility issues. Therefore these attributes aren't recommended.

## HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
    <meta name="viewport" content="width=device-width, initial-scale=1">
    ...
  </head>
  ...
```



# Size content to the viewport

An image has fixed dimensions and if it is larger than the viewport will cause a scrollbar. A common way to deal with this problem is to give all images a max-width of 100%.

This will cause the image to shrink to fit the space it has, should the viewport size be smaller than the image. However because the max-width, rather than the width is 100%, the image will not stretch larger than its natural size. It is generally safe to add the following to your stylesheet so that you will never have a problem with images causing a scrollbar.

## CSS

```
img {  
  max-width: 100%;  
  display: block;  
}
```

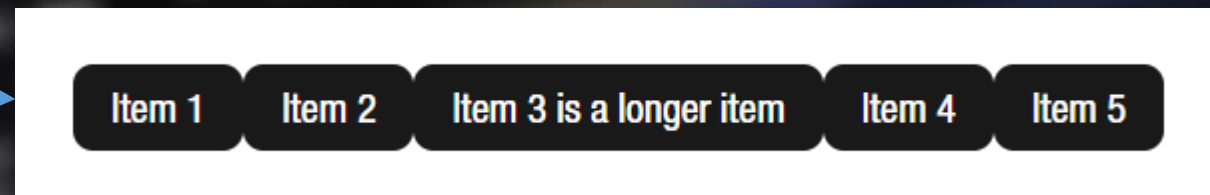
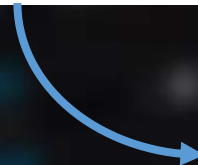
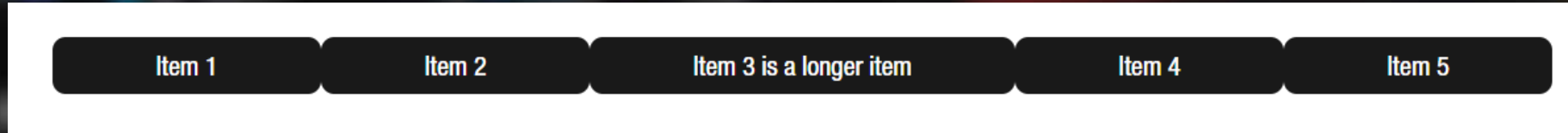
An example of reponsive float element on [GitHub](#).

# Flexbox

Flexbox is a layout method that is ideal when you have a set of items of different sizes and you would like them to fit comfortably in a row or rows, with smaller items taking less space and larger ones getting more space.

## CSS

```
.items {  
  display: flex;  
  justify-content: space-between;  
}
```





# Use CSS media queries for responsiveness

Media queries are simple filters that can be applied to CSS styles. They make it easy to change styles based on the types of device rendering the content, or the features of that device, for example width, height, orientation, ability to hover, and whether the device is being used as a touchscreen.

To provide different styles for printing, you need to target a type of output so you could include a stylesheet with print styles or, alternatively, you could include print styles within your main stylesheet using a media query.

## HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
    <link rel="stylesheet" href="print.css" media="print">
    ...
  </head>
  ...
```

## CSS

```
@media print {
  /* print styles go here */
}
```

# Media queries based on viewport size

Media queries enable you to create a responsive experience where specific styles are applied to small screens, large screens, and anywhere in between.

The feature you can detect here is therefore screen size, and you can test for the following things:

- width (min-width, max-width);
- height (min-height, max-height);
- orientation;
- aspect-ratio.

## HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
    <link rel="stylesheet" href="print.css" media="print">
    ...
  </head>
  ...
```

## CSS

```
@media print {
  /* print styles go here */
}
```

# Media queries based on device capability

Given the range of devices available, we cannot make the assumption that every large device is a regular desktop or laptop computer, or any other one. With some newer additions to the media queries specification we can test for features such as the type of pointer used to interact with the device and whether the user can hover over elements.

- `hover;`
- `pointer;`
- `any-hover;`
- `any-pointer.`

## Testing Media Features for pointer and hover

View this demo using different devices.

Your device allows you to hover. You are using a fine pointer, a mouse or trackpad perhaps?

## Testing Media Features for pointer and hover

View this demo using different devices.

Your device allows you to hover. You are using a fine pointer, a mouse or trackpad perhaps?

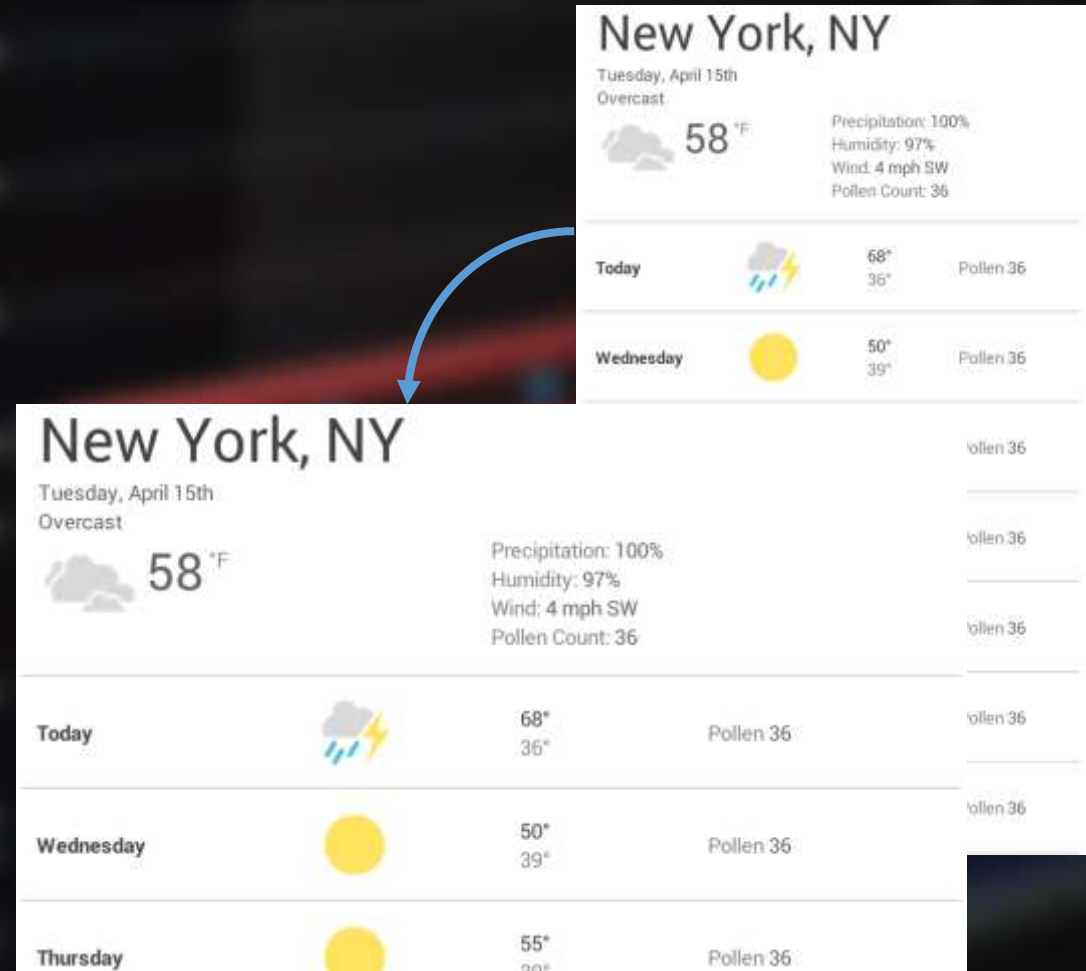


# How to choose breakpoints

A tip: design the content to fit on a small screen size first, then expand the screen until a breakpoint becomes necessary. This allows you to optimize breakpoints based on content and maintain the least number of breakpoints possible.

In the case of Google's weather forecast, the first step is to make the forecast look good on a small screen. Next, resizing the browser until there is too much white space between the elements, you can reorganize the elements in the way showed on the right.

The decision is somewhat always subjective.



# How to choose breakpoints

To insert a breakpoint at 600px, create two media queries at the end of your CSS for the component, one to use when the browser is 600px and below, and one for when it is wider than 600px.

Finally, inside the media query for a max-width of 600px, add the CSS which is only for small screens. Inside the media query for a min-width of 601px add CSS for larger screens.

You have just seen how Google's weather forecast has been coded!

## CSS

```
@media (max-width: 600px) {  
  
}  
  
@media (min-width: 601px) {  
  
}
```

# Pick minor breakpoints when necessary

In addition to choosing major breakpoints when layout changes significantly, it is also helpful to adjust for minor changes. For example, between major breakpoints it may be helpful to adjust the margins or padding on an element, or increase the font size to make it feel more natural in the layout.

As you can see, the first code on the right optimizes the small screen layouts, but going into larger screens it's best to limit to maximum width of the forecast panel so it doesn't consume the whole screen width, in the way exposed inside the second code box here on the right.

## CSS

```
@media (min-width: 360px) {  
  body {  
    font-size: 1.0em;  
  }  
}  
  
@media (min-width: 500px) {  
  .seven-day-fc .temp-low,  
  .seven-day-fc .temp-high {  
    display: inline-block;  
    width: 45%;  
  }  
  
  .seven-day-fc .seven-day-temp {  
    margin-left: 5%;  
  }  
  
  .seven-day-fc .icon {  
    width: 64px;  
    height: 64px;  
  }  
}
```

## CSS

```
@media (min-width: 700px) {  
  .weather-forecast {  
    width: 700px;  
  }  
}
```










# New York, NY

Tuesday, April 15th  
Overcast

 58 °F

Precipitation: 100%  
Humidity: 97%  
Wind: 4 mph SW  
Pollen Count: 36








Today	Wednesday	Thursday	Friday	Saturday	Sunday	Monday
						
68° 36° Pollen 36	50° 39° Pollen 36	55° 39° Pollen 36	54° 43° Pollen 36	64° 46° Pollen 36	64° 50° Pollen 36	61° 50° Pollen 36

## New York, NY

Tuesday, April 15th  
Overcast

 58 °F

Precipitation: 100%  
Humidity: 97%  
Wind: 4 mph SW  
Pollen Count: 36

Today		68° 36°	Pollen 36
Wednesday		50° 39°	Pollen 36
Thursday		55° 39°	Pollen 36
Friday		54° 43°	Pollen 36
Saturday		64° 46°	Pollen 36
Sunday		64° 50°	Pollen 36
Monday		61° 50°	Pollen 36

[Click here](#) to try it by yourself!

**I heard you want to be a web developer**



**Here are a few devices to test your site**