Intersection Observers: Making it easier to lazy load content

What I love about Paul Lewis' Developer Diaries is that he points me to new technologies and better ways to work with web content. In this case (video below) he clued me into a new API: Intersection Observers.

The idea behind Intersection Observers is that we don't really need to load content until it comes into the viewport (it's visible in the browser's window). We can configure the action that happens when the selected object comes into view.

The best example of Intersection Observers I can think of is to lazy load images only when the image in question appears in the viewport, not before. This will make our page load faster because only the two top-most images will load when the page loads and, we all know, images are the biggest hogs when it comes to web page payload.

The script performs the following actions

1. Create the IntersectionObserver and bind it to the function we want it to work with
2. For each image that we want to change

        1. Add the src attribute using the value from the data-src attribute in the same element
        2. Stop observing the current target
    3. Convert node list of all images with data-src attributed to array
    4. Observe each image belonging to the array defined in step 3

```
// Script derived from:
// Quick introduction to the Intersection Observer API
// by Jeremias Menichelli
// 1. Create the IntersectionObserver and bind it to the
// function we want it to work with
let observer = new IntersectionObserver(onChange);

function onChange(changes) {
  // 2. For each image that we want to change
  changes.forEach(change => {
    // * Add the src attribute using the value
    // from the data-src attribute in the same element
    change.target.src = change.target.dataset.src;

    // * Stop observing the current target
    observer.unobserve(change.target);
  })
}

// 5. Convert node list of all images with data-src attributed to array
const imgs = [ ...document.querySelectorAll('img[data-src]') ];

'img[data-src]'// 6. Observe each image belonging to the array above
imgs.forEach(img => observer.observe(img));
```

In the demo page I set the first two images to always load by setting a `src` attribute for the images instead of a `data-src` attribute to be manipulated by the script. This will ensure that the content above the fold or partially above the fold will display regardless of whether the browser supports IntersectionObsservers or not.

Browser support is spotty at best. Acording to caniuse.com only Chrome and Opera support the API out of the box, Firefox supports it behind a flag

(`dom.IntersectionObserver.enabled`) in `about:config` and Edge has it under development. But to load the images in browsers that don't support IntersectionObservers we have to do jump through a few more workarounds.

The idea is that if the browser doesn't support Intersection Observers we load the images right away, using this API as a progressive enhancement.

We modify the script to do as follows:

1. Convert node list of all images with data-src attributed to array
2. Wrap the code on a feature test for IntersectionObserver
3. Create the IntersectionObserver and bind it to the function we want it to work with
4. For each image that we want to change
    1. Add the src attribute using the value from the data-src attribute in the same element
    2. Stop observing the current target
5. Observe each image belonging to the array defined in step 3
6. If the browser doesn't support Interaction Observer then we load all the images right away

```
// 1. Convert node list of all images with
// data-src attribute to an array
const imgs = [ ...document.querySelectorAll('img[data-src]') ];

'img[data-src]'// 2. Wrap the code on a feature test for IntersectionObser
if ('IntersectionObserver' in window) {
  // 3. Create the IntersectionObserver and bind it to the function
  // we want it to work with
  let observer = new IntersectionObserver(onChange);

  function onChange(changes) {
    // 4. For each image that we want to change
    changes.forEach((change) => {
      // * take image url from `data-src` attribute
      change.target.src = change.target.dataset.src;
      // * Stop observing the current target
      observer.unobserve(change.target);
    })
```

```
  }

// 5. Observe each image derived from the array above
  imgs.forEach((img) => observer.observe(img));
} else {
// 6. if the browser doesn't support Intersection Observer
// we log to console and load images manually'Intersection Observers not s
  function loadImages(imgs) {
    imgs.forEach((image) => {
      image.src = image.dataset.src;
    })
  }
  loadImages(imgs);
}

'Intersection Observers not supported'
```

Once this API is deployed on all browsers we'll be able to lazy load content without having to worry about the positioning or threshold of when the images appear in the viewport. Is this the only way to do it? no, it isn't. Brian Rinaldi's Lazy Loading Images on the Web covers how to lazy load images without using Intersection Observers.

- Quick introduction to the Intersection Observer API
- IntersectionObserver's Coming into View
- IntersectionObserver Explainer from WCIG
- IntersectionObserver Spec Work
- Google Developers' Article
- Lazy Loading Images on the Web