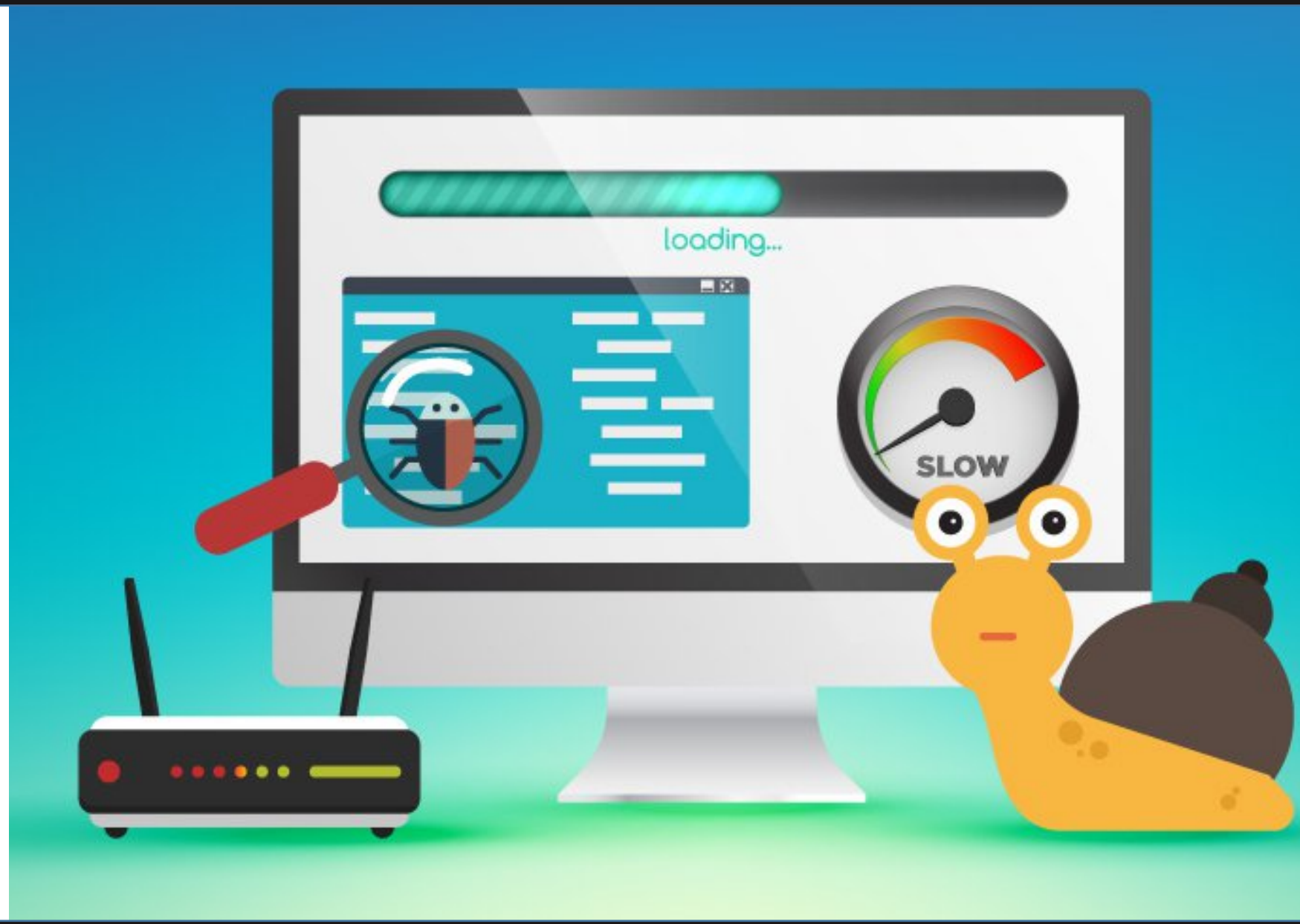
TESTES E PERFORMANCE WEB

TECNOLOGIAS E SISTEMAS DE INFORMAÇÃO PARA A WEB

POLITÉCNICO DO PORTO
ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

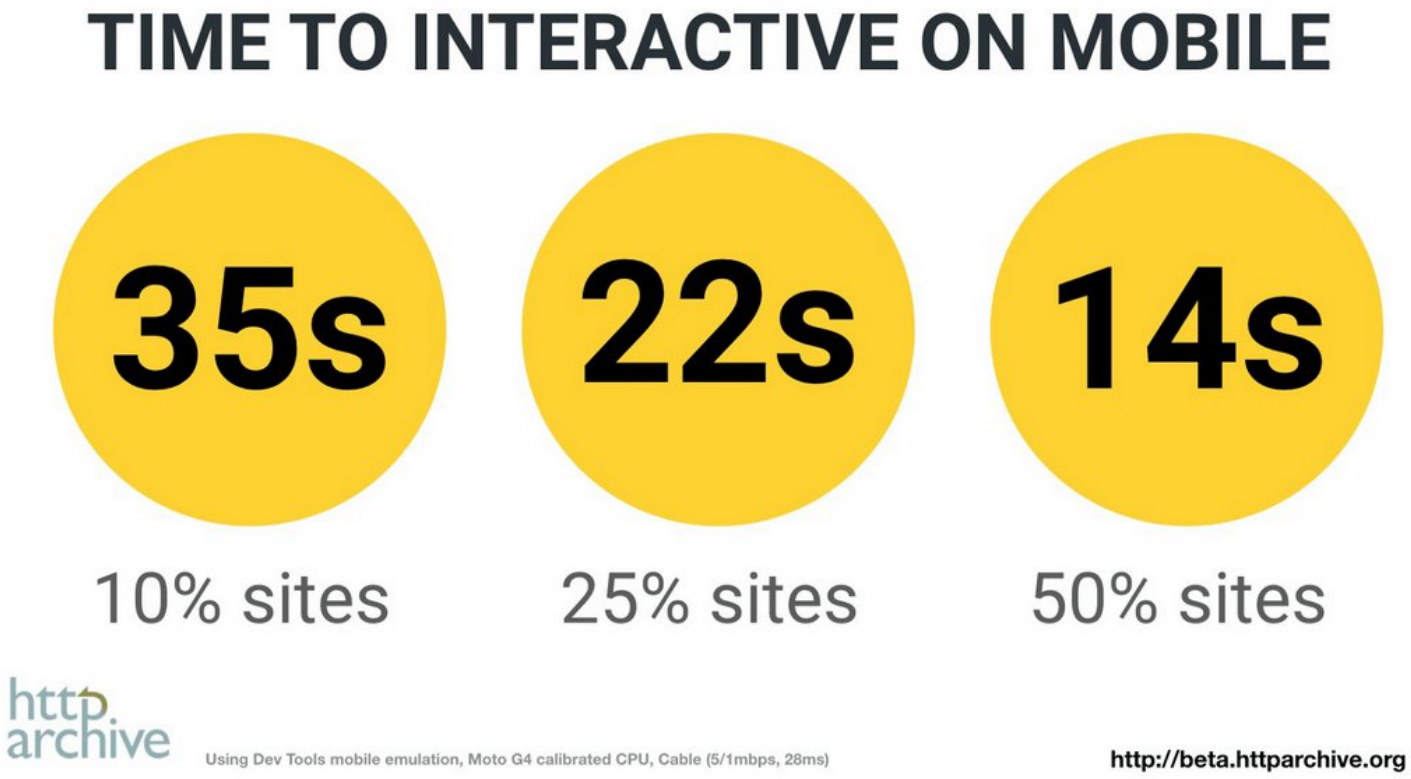
M05 – CODE OPTIMIZATION

TSIW 2023/2024



INTRO

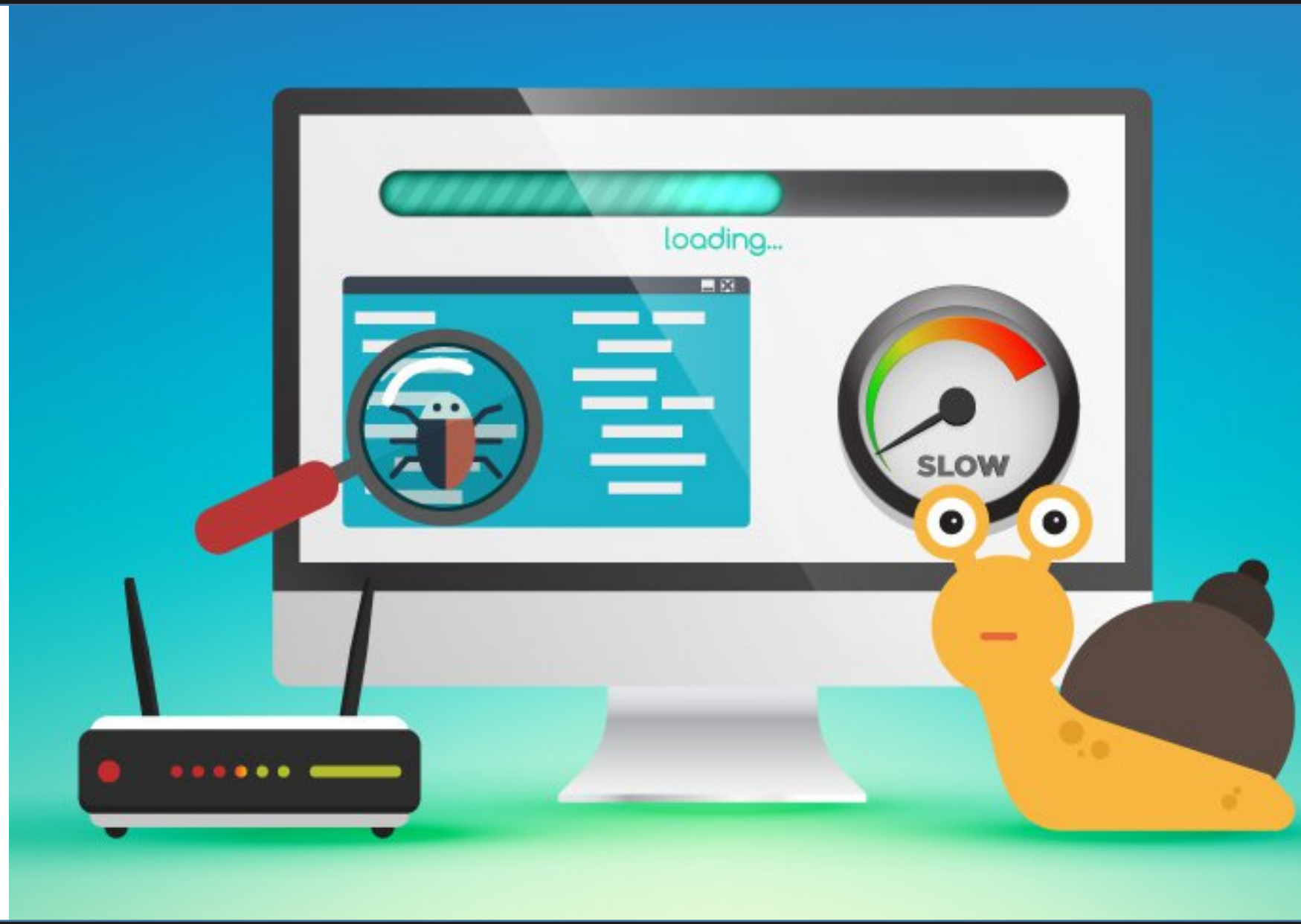
1. Code optimization is one of the most crucial techniques used to decrease Time To Interactive Metric.



AGENDA

1. HTML;
2. CSS;
3. JavaScript.

HTML



1. HTML

- Main Actions
  - **Minify HTML;**
  - Load JavaScript at the end of the page;
  - Preload assets.

1. HTML

Minify HTML:

- By removing comments, white spaces, quotes around attributes, etc...;
- HTML code is smaller than CSS or JavaScript, so performance gains will be less;
- Can be integrated as a CMS plugin, framework module, or build system.

1. HTML

- Main Actions
  - Minify HTML;
  - **Load JavaScript at the end of the page;**
  - Preload assets.

1. HTML

Load JavaScript at the end of the page

- When the browser encounters a <script> tag in the HTML, it halts all other operations while it downloads and parses the code;
- This is known as render-blocking process;
- It is normally more effective to place <script> tag at the bottom of the page before closing the <body> tag;
- This improves performance, since the content is viewable before an attempt is made to process JavaScript.

1. HTML

Load JavaScript at the end of the page

- Two attributes can be added to a <script> tag to ensure JavaScript is loaded in the background without blocking the render process:
  - **defer:** the script is executed when the DOM is ready and shortly before the DOMContentLoaded event. All deferred scripts are run in the order they are referenced on the page;
  - **async:** the script is executed once it has downloaded. This could occur at any point during or after the page has loaded, so it cannot have have other script dependencies.

1. HTML

Load JavaScript at the end of the page

- Deferred scripts run when the DOM is ready, but this can occur before the CSSDOM has been parsed. A script that analyses applied CSS color can fail;
- Scripts placed at the bottom of the page are never affected by this issue.

1. HTML

Load JavaScript at the end of the page

EXAMPLE 1

1. HTML

Load JavaScript at the end of the page

EXAMPLE 2

1. HTML

- Main Actions
  - Minify HTML;
  - Load JavaScript at the end of the page;
  - **Preload assets.**

1. HTML

Preload assets

- The HTML <link> tag has a **preload** attribute;
- This specifies resources the page requires, so download can start immediately rather than waiting for its reference in the HTML;
- For example, a page with a <script> tag just before the closing <body> can be preloaded in the HTML <head>.

```html
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My page</title>

  <!-- preload script -->
  <link rel="preload" href="script.js" as="script" />

  <link rel="stylesheet" href="styles.css" />
</head>
<body>

  <h1>My page</h1>
  <p>Lots of content...</p>

  <!-- load script (may be ready) -->
  <script src="script.js"></script>

</body>
</html>
```

1. HTML

Preload assets

- The larger the HTML page, the greater the preloading benefit;
- The **as** attribute allows browser to prioritize and cache assets more effectively;
- That is, a script is downloading before a video because is more critical to the page's operation.

  Permitted values are:
  - audio;
  - document;
  - embed;
  - fetch;
  - font;
  - image;
  - object;
  - script;
  - style;
  - track;
  - video;
  - worker
- An optional type attribute defines the resource's MIME type, so the browser can make further optimizations to avoid downloading unsupported assets.

```
<link rel="preload" href="video.mp4" as="video" type="video/mp4" />
```

1. HTML

Preload assets

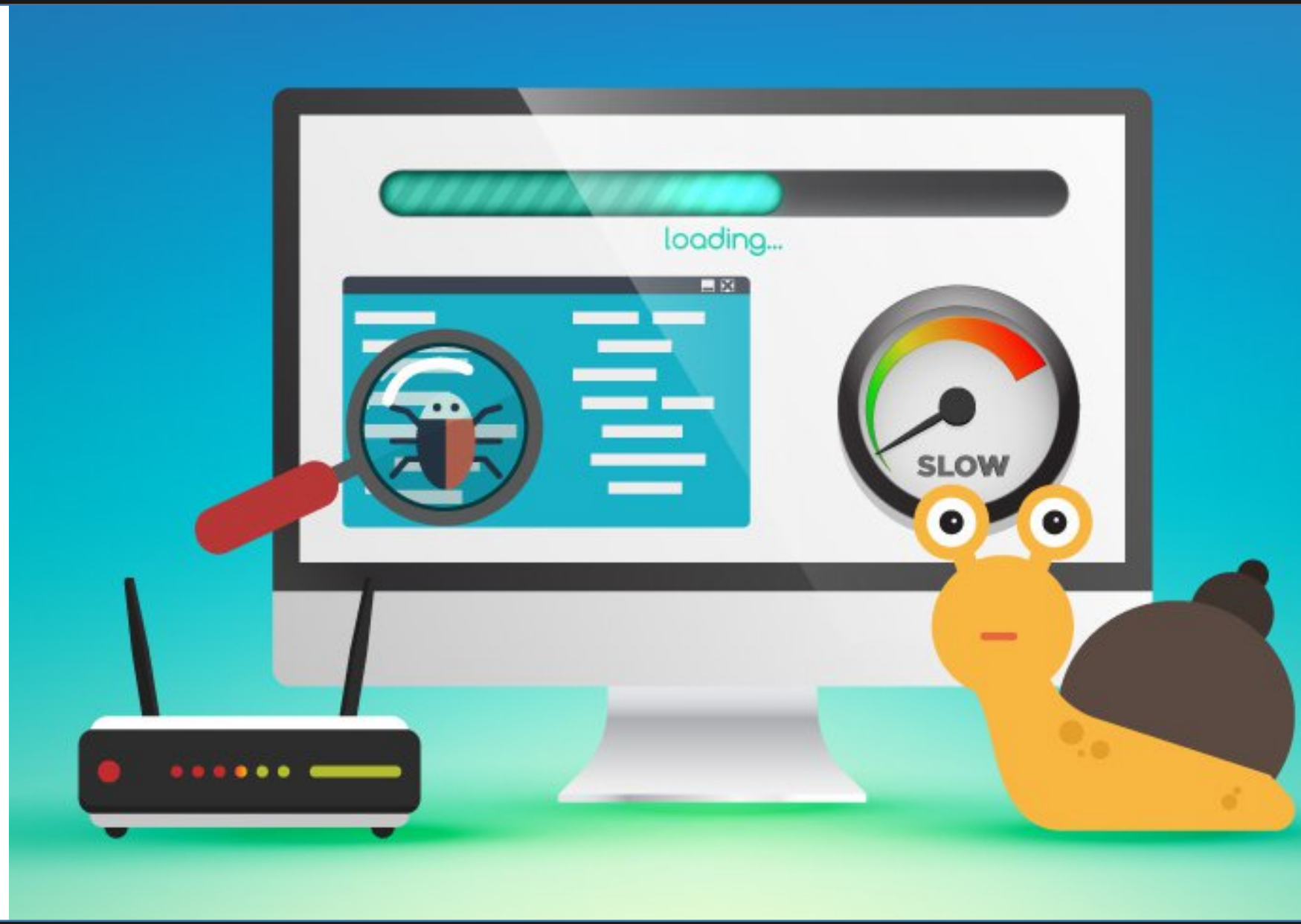
- Other attributes and an API:
  - **prefetch:** similar to preload, except that it is intended to fetch resources that will be in the next navigation/page load. Browsers give prefetch a lower priority;
  - **prerender:** render a specific web page in the background. Since this potentially wastes bandwidth, browsers often limit processing and memory use;
  - **dns-prefetch:** can be used to resolve a domain name to an IP address before resources are requested, while preconnect establishes a connection to a server;
  - **portals:** renders a page in the background and allows the user to instantly navigate to it. The technology is new and yet to be fully implemented in any browser.

1. HTML

Preload assets

EXAMPLE 3

CSS



2. CSS

- Main Actions
  - **Concatenate and minify CSS;**
  - Use modern CSS3 layouts;
  - Remove unused CSS;
  - Embrace CSS3 animations;
  - Avoid animating expensive properties;
  - Indicate which elements will animate.

2. CSS

Concatenate and minify CSS

- Stylesheets can be loading using HTML <link> tags and CSS @import rules;
- Loading separate files is usually inefficient, because each @import blocks the browser's rendering process;
- The imported file could have further nested @import rules;
- Performance can therefore be improved using:
  - Concatenation: all partials are combined into a single large file in the necessary source order;
  - Minification: unnecessary comments, whitespaces, and characters are removed in order to reduce the file size.

2. CSS

Concatenate and minify CSS

- A build process or pre-processor can automate CSS concatenation and minification, but online tools are also available:
  - CSS Minifier;
  - Minifier.org;
  - CSS Compressor;
  - CSS Minify;
  - Online Compressor.

2. CSS

Concatenate and minify CSS

EXAMPLE 4

2. CSS

- Main Actions
  - Concatenate and minify CSS;
  - **Use modern CSS3 layouts;**
  - Remove unused CSS;
  - Embrace CSS3 animations;
  - Avoid animating expensive properties;
  - Indicate which elements will animate.

2. CSS

Use modern CSS3 layouts

- **Flexbox** - for 1D layouts, which (can) wrap to the next row according to the widths of each block. It is ideal for menus, image galleries, cards, etc...;
- **Grid** - for 2D layouts with explicit rows and columns. It is ideal for page layouts;
- Both options are similar to develop, use far less code, can adapt to any screen size, can remove the need for media queries, and render faster than floats because the browser can natively determine an optimum layout.

2. CSS

- Main Actions
  - Concatenate and minify CSS;
  - Use modern CSS3 layouts;
  - **Remove unused CSS;**
  - Embrace CSS3 animations;
  - Avoid animating expensive properties;
  - Indicate which elements will animate.

2. CSS

Remove unused CSS

- The smaller your stylesheet, the quicker it will download, the sooner it will parse, and the faster your page will become;
- We all start with good intentions, but CSS can bloat over time as the number of feature increases;
- It is easier to retain old, unnecessary code that remove it and risk breaking something;
- Those using a CSS framework such as bootstrap may find they are only using a small fraction of the code.

2. CSS

Remove unused CSS

- **Recommendations:**
  - Organize CSS into smaller files (partials) with clear responsibilities (which can be concatenated into a single file at build time);
  - It is easier to remove a carousel widget if the CSS is clearly defined in widgets/_carousel.css;
  - Consider naming methodologies such as BEM (Block Element Modifier) to aid the development of discrete components;
  - Avoid deeply nested SASS/Pre-Processor declarations. The expanded code can become unexpectedly large;
  - Avoid using !important to override the cascade, and inline styles in HTML.

2. CSS

Remove unused CSS

- Chrome's coverage panel helps you to locate unused CSS/JS code;
  - Select Coverage from Chrome DevTools more tools sub-menu;
  - Hit the record button;
  - Browser your app;
  - Click any file to open its source;
  - Unused code is highlighted in red at line number gutter.

```
716 #menu .open ul {
717   display: block
718 }
719
720 #menu a,#menu strong {
721   display: block;
722   font-size: 1.2em;
723   width: 9em;
724   padding: .2em 0 .4em 1.8rem;
725   margin: 0
726 }
727
728 #menu ul a,#menu ul strong {
729   padding-left: 2.8rem
730 }
731
732 #menu a.opener {
733   float: left;
734   width: 0;
735   padding-left: 1.8rem;
736   padding-right: 0;
737   background: url("data:image/svg+xm
738   background-size: 30%;
739   transform: rotate(0deg);
740   transition: transform .2s ease-i
741   white-space: nowrap;
742   overflow: hidden
743 }
744
745 #menu .open .opener {
746   transform: rotate(90deg)
747 }
```

2. CSS

Remove unused CSS

- The following tools provide options to analyse HTML and CSS usage either at build time or by crawling URL's so that redundant code can be identified:
  - PurifyCSS (there is also an online version);
  - PurgeCSS;
  - UnCSS;
  - UnusedCSS.

2. CSS

- Main Actions
  - Concatenate and minify CSS;
  - Use modern CSS3 layouts;
  - Remove unused CSS;
  - **Embrace CSS3 animations;**
  - Avoid animating expensive properties;
  - Indicate which elements will animate.

2. CSS

Embrace CSS3 animations

- Native CSS3 transitions and animations will always be faster and require less code than JavaScript-powered equivalents;
- It should not be necessary add a library for a typical fade, show, hide, move effects;
- Very old browsers may not support the properties, but CSS degrades gracefully, and users will rarely know they are missing anything;
- JS animations should only be considered when fine-grained control is required (e.g. HTML5 games, interactive chats, <canvas> manipulation.

2. CSS

- Main Actions
  - Concatenate and minify CSS;
  - Use modern CSS3 layouts;
  - Remove unused CSS;
  - Embrace CSS3 animations;
  - **Avoid animating expensive properties;**
  - Indicate which elements will animate.

2. CSS

Avoid animating expense properties

- Once the browser has parsed the HTML document and styles, it renders elements in three stages:
  - **Layout:** the calculation of how much space an element requires and how it affects elements around it;
  - **Paint:** the filling of pixels with color;
  - **Composite:** the drawing of layers in the correct order when they overlap.

2. CSS

Avoid animating expense properties

- Animating the dimensions or position of an element can cause the whole page to re-layout on every frame;
- Performance can therefore be improved if an animation only affects the compositing stage;
- The most efficient animations only uses:
  - Opacity and/or;
  - Transform or translate (move), scale, skew, or rotate an element (the original space the element is not used and is not altered so the layout is not affected).

2. CSS

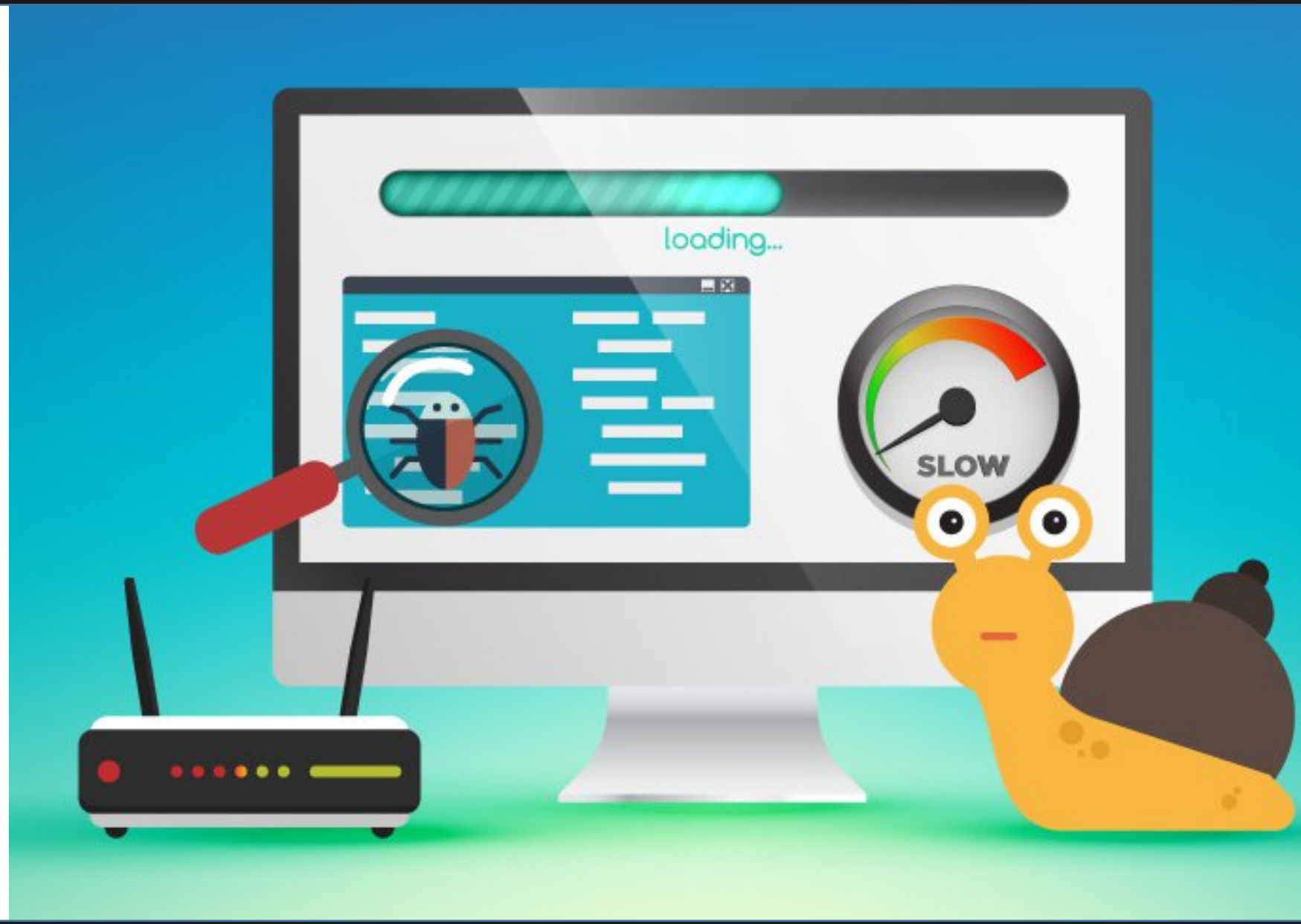
- Main Actions
  - Concatenate and minify CSS;
  - Use modern CSS3 layouts;
  - Remove unused CSS;
  - Embrace CSS3 animations;
  - Avoid animating expensive properties;
  - **Indicate which elements will animate.**

2. CSS

Indicate which elements will animate

- The will-change property allows CSS developers to indicate how an element will be animated so the browser can make performance optimizations in advance;
- Any number of comma-separated properties can be defined;
- However:
  - Only use will-change as last resource to fix animation issues;
  - It should not be used to anticipate performance issues;
  - Don’t apply it to too many elements;
  - Give it sufficient time to work. Don’t begin animations immediately.

JAVASCRIPT



3. JAVASCRIPT

- Main Actions
  - **Variables;**
  - Loops;
  - DOM;
  - Events;
  - Remove unused code;
  - Concatenate and minify JavaScript.

3. JAVASCRIPT

Variables

- Try not to use global variables:
  - The scripting engine needs to look through the scope, when referencing global variables from within function or another scope, the variable will be destroyed when the local scope is lost;
  - If variables in global scope cannot persist through the lifetime of the script, the performance will be improved.

3. JAVASCRIPT

Variables

- Avoid unnecessary variables:
  - Don’t create new variables if you don’t plan to save values;
  - Often you can replace code like this:

```
// BAD
const fullName = `${firstName} ${lastName}`;
document.getElementById('demo').innerHTML = fullName;

// GOOD
document.getElementById('demo').innerHTML = `${firstName} ${lastName}`;
```

3. JAVASCRIPT

- Main Actions
  - Variables;
  - **Loops;**
  - DOM;
  - Events;
  - Remove unused code;
  - Concatenate and minify JavaScript.

3. JAVASCRIPT

Loops

- Don’t use nested loops if not required;
- Reduce activity in loops:
  - Each statement in a loop, including the for statement, is executed for each iteration of the loop;
  - Statements or assignments that can be placed outside the loop will make the loop run faster.

```
// BAD
for (let i = 0; i < arr.length; i++) {
    // ...
}

// GOOD
const len = arr.length;
for (let i = 0; i < len; i++) {
    // ...
}
```

3. JAVASCRIPT

- Main Actions
  - Variables;
  - Loops;
  - **DOM;**
  - Events;
  - Remove unused code;
  - Concatenate and minify JavaScript.

3. JAVASCRIPT

DOM

- Cache regularly used nodes:
  - Regularly used DOM nodes should be stored as JavaScript variables so they don't need to be re-fetched;
  - The DOM references are retained even when other tree nodes are modified.

```
const
    main = document.getElementsByTagName('main')[0],
    heading = main.querySelector('h1'),
    tables = main.getElementsByTagName('table');
```

3. JAVASCRIPT

DOM

- Cache regularly used nodes:
  - querySelector() and querySelectorAll() can find elements using JQuery-like CSS selectors;
  - They are slower than getElementById(), getElementByTagName() and getElementByClassName(), although the speed difference is unlikely to affect most applications;
  - getElementByTagName() and getElementByClassName() also return live HTMLCollections, which update automatically as the DOM is modified. Thus it is not necessary to rerun the query.

```
const
    main = document.getElementsByTagName('main')[0],
    heading = main.querySelector('h1'),
    tables = main.getElementsByTagName('table');
```

3. JAVASCRIPT

DOM

- Minimize reflows:
  - When an element is added, modified, or removed from a page, it can trigger a cascade of layout changes to surrounding elements;
  - Some techniques:
    - Use opacity and/or transform to translate (move), scale, or rotate the element;
    - Limit the scope of the reflow by changing elements low in the DOM tree (those without deeply nested children);
    - Update elements in their own position: absolute, or position: fixed, layer;
    - Modify hidden elements (display:none), then show them after the change has been applied.

3. JAVASCRIPT

DOM

- Batch-update styles;
- The next example could cause three reflows:

```
let myelement = document.getElementById('myelement');
myelement.width = '100px';
myelement.height = '200px';
myelement.style.margin = '10px';
```

- Performance can be improved by appending a class;

```
let myelement = document.getElementById('myelement');
myelement.classList.add('newstyles');
```

```
.newstyles {
    width: 100px;
    height: 200px;
    margin: 10px;
}
```

- This applies CSS properties in one reflow operation.

3. JAVASCRIPT

DOM

- Batch-update elements:
  - Minimize the number of interactions with DOM;
  - Use DocumentFragment to build elements in memory before applying those changes to page;
  - For example, you can create an unordered list with three items like the image;
  - The DOM is only modified on the last line.

```
// create list
let
    frag = document.createDocumentFragment(),
    ul = frag.appendChild(document.createElement('ul'));

for (let i = 1; i <= 3; i++) {
    let li = ul.appendChild(document.createElement('li'));
    li.textContent = 'item ' + i;
}

// append list to the DOM
document.body.appendChild(frag);
```

3. JAVASCRIPT

DOM

- Reduce DOM access:
  - Accessing DOM is very slow, compared to other JS statements;
  - If you expect to access a DOM element, several times, access it once, and use it as a local variable.

```
// BAD
document.getElementById('demo').innerHTML = 'Hello';
document.getElementById('demo').style.color = 'red';

// GOOD
const obj = document.getElementById('demo');
obj.innerHTML = 'Hello';
obj.style.color = 'red'
```

3. JAVASCRIPT

DOM

- Reduce DOM size:
  - Keep the number of elements in the HTML DOM small;
  - This improve page loading, and speed up rendering (page display), especially on mobile devices;
  - Every attempt to search DOM (e.g. getElementsByTagName) will benefit from a smaller DOM.

3. JAVASCRIPT

- Main Actions
  - Variables;
  - Loops;
  - DOM;
  - **Events;**
  - Remove unused code;
  - Concatenate and minify JavaScript.

3. JAVASCRIPT

Events

- Bind events sparingly:
  - Applications can have dozens of event handlers;
  - A handler function is registered to an event when it is triggered on a specific DOM element;

```
myElement.addEventListener('click', doSomething);
```

  - Each bound event has a performance hit. Ideally you should:
    - Only add events you required;
    - Return from handler functions quickly;
    - Unbind using removeEventListener when an event is no long necessary.

3. JAVASCRIPT

Events

- Bind events sparingly:
  - Make use of event delegation;
  - In a HTML <table> with thousands of cells to react to a <td> being clicked attaching an event to each cell requires significant processing and would need to be reapplied if the table change;
  - Instead, attach a single event handler to <table> and examine the target.

```
myElement.addEventListener('click', doSomething);

// handle a click on any <td> element
document.getElementById('mytable').addEventListener('click', (e) => {

    let t = e.target.closest('td');
    if (!t) return;

    console.log('clicked cell', t);

});
```

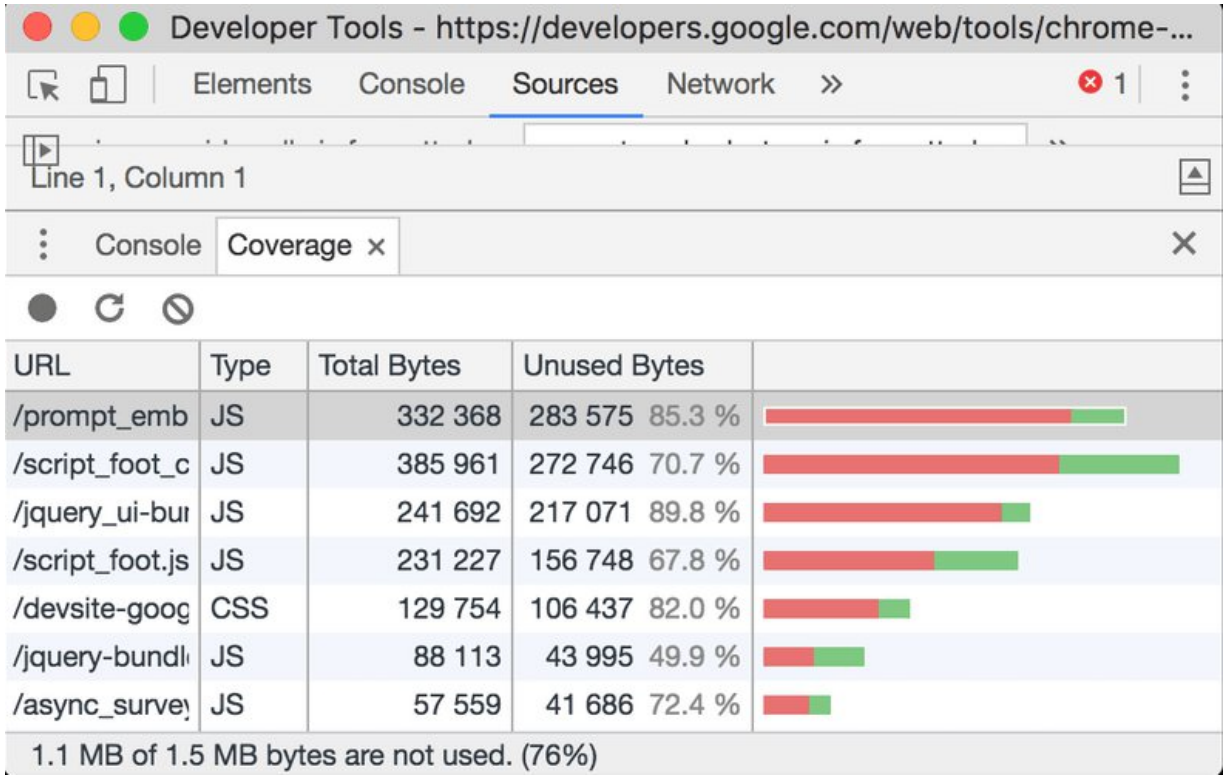
3. JAVASCRIPT

- Main Actions
  - Variables;
  - Loops;
  - DOM;
  - Events;
  - **Remove unused code;**
  - Concatenate and minify JavaScript.

3. JAVASCRIPT

Remove unused code

- The coverage tab in DevTools will also tell you how much CSS and JS code in your application is unused.



3. JAVASCRIPT

- Main Actions
  - Variables;
  - Loops;
  - DOM;
  - Events;
  - Remove unused code;
  - **Concatenate and minify JavaScript.**

3. JAVASCRIPT

Concatenate and minify JavaScript

- Code is split into multiple file with related or self-contained functionality;
- This makes development more practical:
  - Files are easier to understand;
  - Each can be tested individually;
  - Reuse in other projects is easier.
- Dependencies are declared in a way where Script A is loaded before Script B;
- Multiple JavaScript files can be loaded in a single web page using:
  - More than one HTML <script> element defined in dependency order;
  - ES6 modules, which import dependencies when they are required within a script.

3. JAVASCRIPT

Concatenate and minify JavaScript

- **Minification** is the process of create a smaller but perfectly valid code file by:
  - Removing whitespace;
  - Any code that is not necessary.
- **Terser** is a popular JS compression tool and webpack V4 includes a plugin for this library by default to create minified build files:
  - If using webpack V4 or greater, you should be good to go without doing any additional work.

3. JAVASCRIPT

Concatenate and minify JavaScript

- Alternatively, the process can be handled manually to improve performance on an existing site:
  - JSCompress;
  - minifier.org;
  - Minify your JavaScript;
  - Online Compressor;
  - Packer.

3. JAVASCRIPT

Concatenate and minify JavaScript

EXAMPLE 5