**Frontend Challenges**

**Question 1**

1 . Colors obf the elements   
 a. FirstDiv will be red, because it has the red-card class applied to it. The general div rule sets the background to green, but the class red-card overrides that and sets it to red.

b. Second Div will be orange, because it has the red-card class applied to it,but the more specific #secondDiv rule in the stylesheet overrides both the div rule and the red-card class, setting its background color to orange.

2. document.getElementById('firstDiv').style.background = 'pink';

3. document.getElementById('secondDiv').classList.add('yellow-card');

**Question 2**  
  
1. The function call compareIt(5, "5") returns true because the comparison uses the loose equality operator (==). In JavaScript, == allows type coercion, meaning it attempts to convert the operands to the same type before comparing them. In this case, the string "5" is converted to the number 5, and since the values are the same, the comparison returns true.

2. To check both the values and data types, you should use the strict equality operator (===), which doesn't perform type coercion. It returns true only if the values and their data types are the same.

function compareIt(num1, num2) {

return num1 === num2;

}

With this change, compareIt(5, "5") will now return false because 5 (a number) and "5" (a string) are of different types.

We can aslo introduce Typescript, which is a superset of javascript that adds static typing. Example

function compareIt(num1: number , num2: number): boolean {

return num1 === num2;

}

**Question 3**

**1. Responsive design is achieved using flexible layouts, images, and CSS media queries.**   
Examples: Please refer to the styles.css file.  
  
Key Principles of Responsive Design

* **Fluid Grids**: Instead of fixed-width layouts, use fluid grids where elements resize in proportion to the viewport. This flexibility allows layouts to adapt seamlessly across devices.
* **Media Queries**: These are CSS techniques that apply styles based on device characteristics, primarily screen width. This allows you to change styles based on specific conditions.
* **Flexible Images**: Images should be able to scale within their containing elements, maintaining their aspect ratio while ensuring they don’t overflow their containers.
* **Mobile-First Approach**: Designing for mobile devices first and then progressively enhancing the design for larger screens is often more efficient and effective.

**2. Benefits of Bundling .js Scripts into One File**

* Reduced HTTP Requests: Bundling minimizes the number of requests to the server, improving load times. Each JavaScript file requires a separate HTTP request to the server. When a web page has multiple JavaScript files, it can lead to increased load times due to the overhead of establishing multiple connections.
* Improved Performance: Bundling allows for better caching strategies. When users revisit a website, the browser can cache the bundled file, avoiding the need to download it again. This is particularly beneficial for returning visitors, as the bundled file can be loaded from cache rather than fetched from the server.
* Cache Optimization: A single bundled file can be cached by the browser, reducing load times for repeat visits.
* Minification: Bundling tools often include minification, which reduces file size by removing unnecessary characters from code base like whitespace and comments.

**3. Ensuring the Browser Understands Sass Styling**

Sass (Syntactically Awesome Style Sheets) is a CSS preprocessor that extends CSS with features like variables, nested rules, mixins, and functions. These features enhance the organization and maintainability of stylesheets, allowing developers to write more structured and reusable code.

1. Esure your HTML link to the compiled CSS file  
   Example: <link rel="stylesheet" href="dist/styles/main.css">
2. Use a Build Tool: Use tools like Webpack, Gulp, or Parcel to compile Sass files into CSS.  
   Example: "scripts": { "sass": "node-sass src/styles/main.scss dist/styles/main.css" }
3. Use a Build Tool: Use tools like Webpack, Gulp, or Parcel to compile Sass files into CSS.  
   Example: using NodeJS = npm install node-sass --save-dev  
   “scripts”: {  
    “saas”: “node-saas src/styles/main.scss dist/styes/main.css”  
   }

**4. Ensuring Browser Compatibility with Newer JavaScript Features (ES6/7)**

* Use Transpilers: Utilize Babel to transpile ES6/7 code to ES5, which is supported by most browsers.
* Polyfills: Include polyfills for newer features that aren't available in older browsers.