Web Development: Comprehensive Teaching Notes

Antigravity AI Assistant

December 20, 2025

Contents

1 Chapter 1: HTML - HyperText Markup Language

This document provides comprehensive teaching notes for HTML, covering everything from basic structure to advanced semantic elements and media integration.

Exam Tips & Hints

- **Semantic Tags**: Always use semantic tags like `<header>`, `<nav>`, and `<footer>` instead of just `<div>`. This is a frequent exam question.
- **Tables**: Practice `rowspan` and `colspan` thoroughly. Designers often test your ability to structure complex tables from a visual diagram.
- **Attributes**: Remember that `alt` is mandatory for accessibility in `<img>` tags.

1.1 Introduction to HTML

HTML stands for **HyperText Markup Language**. It was created by Tim Berners-Lee in 1989 to distribute information across a network of computers.

- **Hypertext**: Documents contain links that allow jumping to other places or documents.
- **Markup**: Uses tags and attributes to define the structure and presentation of data.
- **Interpretation**: Directly interpreted by the web browser.
- **Case Sensitivity**: HTML is **not** case-sensitive. Multiple spaces are ignored.

1.2 Basic Construction of an HTML Page

Every HTML5 document should follow this basic structure:

Practical Example: Basic HTML Page

A simple page demonstrating headings, lists, and links:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>My First Web Page</title>
</head>
<body>
    <h1>Welcome to My Portfolo</h1>
    <p>This is a small demonstration of <strong>HTML fundamentals</
       strong>.</p>

    <h2>My Skills</h2>
    <ul>
        <li>HTML5 (Structure)</li>
        <li>CSS3 (Design)</li>
        <li>JavaScript (Interactivity)</li>
    </ul>

    <h2>Projects</h2>
    <ol>
        <li><a href="https://github.com/user/project1">Weather App</a><
           /li>
        <li><a href="project2.html">Personal Blog</a></li>
    </ol>
</body>
</html>
```

1.2.1 Key tags in the structure:

- `<!DOCTYPE html>`: Tells the browser to expect HTML5.
- `<html>`: The root element that wraps all content.
- `<head>`: Contains metadata (title, character set, links to external files) not visible on the page.
- `<body>`: Contains the visible content of the web page.

1.3 Elements, Tags, and Attributes

- **Element**: Defined by a start tag, content, and an end tag (e.g., `<p>Text</p>`).
- **Tag**: The angle brackets surrounding an element name. Most occur in pairs (opening and closing).
- **Attribute**: Provides additional information about an element (e.g., `id`, `class`, `src`, `href`). Defined within the opening tag.

1.4 Text Formatting and Headings

1.4.1 Headings

HTML provides six levels of headings from `<h1>` (most important) to `<h6>` (least important). Search engines use these to understand page hierarchy.

1.4.2 Text Elements

- `<p>`: Paragraph.
- `<b>` or `<strong>`: Bold/Strong importance.
- `<i>` or `<em>`: Italic/Emphasized text.
- `<mark>`: Highlighted text.
- `<small>`: Smaller text.
- `<strike>`: Strikethrough text.
- `<u>` or `<ins>`: Underlined/Inserted text.
- `<sub>` and `<sup>`: Subscript and Superscript.

1.4.3 Special Characters

Use entities for symbols not on the keyboard:

- ` `: Non-breaking space
- `&copy;`: ©
- `&reg;`: ®
- `&euro;`: €
- `&gt;`: >

1.5 Lists

1.5.1 Unordered Lists (`<ul>`)

Used for items where order doesn't matter.

- Each item is wrapped in `<li>`.
- Attributes: `type` (circle, square, disc).

1.5.2 Ordered Lists (`<ol>`)

Used for sequential data.

- Attributes: `type` (1, A, a, I, i), `start`, `reversed`.

1.6 Links and Navigation

1.6.1 Types of Links

- **Internal Links**: Link to another page in the same website using a **relative path**.
- **External Links**: Link to a different website using an **absolute path** (starting with `http://` or `https://`).

1.6.2 Implementation

```
<a href="destination.html" target="_blank">Clickable Text</a>
```

- `href`: The destination URL.
- `target="_blank"`: Opens the link in a new tab.

1.7 Media: Images, Video, and Audio

Image Maps: Allow multiple links (hotspots) on a single image using `<map>` and `<area>`.

Practical Example: Image Maps

Defining clickable regions on a local image:

```
<img src="world-map.jpg" alt="World Map" usemap="#mapname">

<map name="mapname">
  <!-- Circular hotspot -->
  <area shape="circle" coords="100,100,50" href="europe.html" alt="
     Europe">
  <!-- Rectangular hotspot -->
  <area shape="rect" coords="200,50,400,150" href="asia.html" alt="Asia
     ">
</map>
```

1.7.1 Video and Audio (HTML5)

```
<video controls autoplay loop muted>
  <source src="movie.mp4" type="video/mp4">
</video>

<audio controls>
  <source src="song.mp3" type="audio/mpeg">
</audio>
```

Attributes: `controls`, `autoplay`, `muted`, `loop`.

1.8 HTML Tables

Used to display data in a grid (rows and columns).

- `<table>`: Starts the table.
- `<tr>`: Table Row.
- `<td>`: Table Data (cell).
- `<th>`: Table Header cell (bold and centered by default). `<caption>`: Adds a title to the table.

`rowspan` and `colspan`: Allow cells to span multiple rows or columns.

Practical Example: Advanced Table Layout

A complex schedule table using spanning:

```
<table border="1">
  <tr>
    <th rowspan="2">Day</th>
    <th colspan="2">Activities</th>
  </tr>
  <tr>
    <th>Morning</th>
    <th>Evening</th>
  </tr>
  <tr>
    <td>Monday</td>
    <td>Coding</td>
    <td>Gym</td>
  </tr>
  <tr>
    <td>Tuesday</td>
    <td colspan="2">Project Deadline (Full Day)</td>
  </tr>
</table>
```

1.9 HTML Forms

Used to collect user input and send it to a server.

- `<form>`: Wraps the form elements.
  - `action`: URL where data is sent.
  - `method`: `GET` (visible in URL) or `POST` (secure/hidden).
- **Common Elements**:
  - `<input>`: versatile input field (type="text", "password", "submit", etc.).
  - **HTML5 Enhancements**: `type="email"`, `type="date"`, `type="number"`, `type="color"`. These provide built-in validation.

1.10 Semantic HTML and Accessibility

Semantic HTML gives meaning to code, helping search engines and screen readers.

1.10.1 Block vs. Inline

- **Block-level**: Starts on a new line, takes full width (e.g., `<div>`, `<h1>`, `<p>`, `<ul>`).
- **Inline-level**: Stays in the flow, only takes necessary width (e.g., `<span>`, `<a>`, `<img>`).

Comprehensive Exam Prep: HTML

Past Paper Questions (Sessional & Final)

1. **MCQ:** Which HTML attribute is used to define inline styles? *(Sessional 1 2025)*

   a. style
   b. class
   c. styles
   d. font

   Answer: a) style

2. **MCQ:** What is the correct HTML element for the largest heading? *(Final 2024)*

   a. `<h1>`
   b. `<h6>`
   c. `<head>`
   d. `<header>`

   Answer: a) `<h1>`

3. **MCQ:** Which HTML attribute is used to provide a unique identifier for an element? **Answer:** a) id

4. **MCQ:** What does the `<ol>` tag represent in HTML? **Answer:** a) Ordered List

5. **Fill in the blank:** To make a hyperlink open in a new browser tab, the attribute of the `<a>` tag is **target** and should be set to **_blank**.

Extra Practice Questions

Multiple Choice Questions

1. Which tag is used to create a drop-down list?

   a. `<input type="dropdown">`

   b. `<list>`

   c. `<select>`

   d. `<option>`

   Answer: c) `<select>`

2. Which semantic tag determines the footer of a document or section?

   a. `<bottom>`

   b. `<footer>`

   c. `<section>`

   d. `<aside>`

   Answer: b) `<footer>`

Code Analysis

Question: Write the HTML code to create a form that sends data to `/submit` using the `POST` method. It should contain a text field for "Name" and a submit button.

```
<form action="/submit" method="POST">
    <label for="name">Name:</label>
    <input type="text" id="name" name="user_name">
    <button type="submit">Submit</button>
</form>
```

Theory Short Questions

- **Difference between Block and Inline elements:**
  - **Block**: Starts on a new line and takes up full width (e.g., `div`, `p`, `h1`).
  - **Inline**: Starts on the same line and takes only necessary width (e.g., `span`, `a`, `img`).
- **Purpose of the `alt` attribute:**
  Provides alternative text for screen readers (accessibility) and displays if the image fails to load.

2 Chapter 2: CSS - Cascading Style Sheets

Exam Tips & Hints

- **Specificity**: High-probability topic. Remember: Inline Styles > ID > Class/Attribute > Element.
- **Box Model**: Understand that `padding` is inside the border and `margin` is outside.
- **Selectors**: Be comfortable with pseudo-classes like `:hover` and `:nth-child()`.

This document covers the fundamentals and advanced properties of CSS, used to control the visual presentation of web pages.

2.1 What is CSS?

CSS stands for **Cascading Style Sheets**.

- **History**: Created by Hakon Lie in 1994; now a W3C standard.
- **Purpose**: Controls layout, enforces uniformity, saves time, and enables multiple device compatibility.
- **Rules**: A CSS rule consists of a **Selector** and a **Declaration Block**.
  - `selector { property:  value; }`

2.2 Inserting CSS

There are three ways to apply CSS to an HTML document:

1. **Inline Styles**: Added directly to an element using the `style` attribute.
   - Example: `<h1 style="color:red;">Title</h1>`
2. **Internal/Embedded Styles**: Defined inside a `<style>` tag within the `<head>` section.
3. **External Style Sheets**: Defined in a separate `.css` file and linked in the `<head>`.
   - Example: `<link rel="stylesheet" type="text/css" href="mystyle.css">`

Cascading Order: Inline styles have the highest priority, followed by internal and external style sheets.

Attribute Selector: Styles elements based on their attributes.

Practical Example: Specific Selectors

Different ways to target elements:

```
/* Tag: all paragraphs */
p { font-family: Arial; }

/* ID: unique header */
#main-header { background-color: navy; color: white; }

/* Class: reusable button */
.btn-submit { border-radius: 5px; cursor: pointer; }

/* Attribute: target specific inputs */
input[type="text"] { border: 1px solid gray; }
```

Margin: Transparent area outside the border (space between elements).

Practical Example: Box Model Visualization

How properties combine to create the total size:

```
.box {
  width: 200px;     /* Content width */
  padding: 20px;    /* 20px on all sides */
  border: 5px solid black;
  margin: 15px;     /* Space outside the border */
}

/* Total width = 200 + 20(left) + 20(right) + 5(left border) + 5(right
   border) = 250px */
```

2.3 Visual Properties

2.3.1 Colors

Colors can be specified by:

- **Name**: `Tomato`, `DodgerBlue`.
- **RGB**: `rgb(255, 99, 71)`.
- **HEX**: `#ff6347`.

2.3.2 Fonts and Text

- **Font**: `font-family`, `font-size`, `font-weight`.
- **Text**: `text-align`, `text-decoration`, `text-transform`, `color`.

2.3.3 Backgrounds

- `background-color`, `background-image`, `background-repeat` (repeat-x, repeat-y, no-repeat).
- `background-attachment` (fixed, scroll).
- `background-position` (e.g., `right top`).
- **Shorthand**: `background: #ffffff url("img.png") no-repeat right top;`

2.4 Advanced Styling

2.4.1 Links

Links can be styled based on their state:

- `a:link`, `a:visited`, `a:hover`, `a:active`.

2.4.2 Borders and Outlines

- **Borders**: `border-style` (solid, dashed, dotted), `border-width`, `border-color`.
- **Outline**: A line drawn around elements, outside the borders, to make the element "stand out".

Comprehensive Exam Prep: CSS

Past Paper Questions (Sessional & Final)

1. **MCQ:** Which CSS property is used to change text color?

   a. font-style

   b. color

   c. text-color

   d. background

   Answer: b) color

2. **MCQ:** In CSS, which of the following selectors has the highest specificity?

   a. .card

   b. div p

   c. #header

   d. p

   Answer: c) #header (IDs have higher specificity than classes or tags).

3. **Snippet:** Write CSS to style `.btn-primary` with navy background, white text, 10px 20px padding, and no border.

```
.btn-primary {
    background-color: navy;
    color: white;
    padding: 10px 20px;
    border: none;
}
```

Extra Practice Questions

Multiple Choice Questions

4. How do you make a list that lists its items with squares?

    a. `list: square;`

    b. `list-type: square;`

    c. `list-style-type: square;`

    d. `list-style-image: square;`

    Answer: c) `list-style-type: square;`

5. What is the default value of the `position` property?

    a. relative

    b. fixed

    c. absolute

    d. static

    Answer: d) static

Advanced Styling Concepts

Question: Explain the CSS Box Model with a diagrammatic description.
Answer: The Box Model consists of four layers surrounding the HTML element:

- **Content**: The actual text or image.
- **Padding**: Transparent area between content and border.
- **Border**: A line going around the padding and content.
- **Margin**: Transparent area outside the border, separating the element from others.

Question: Center a `div` horizontally and vertically using Flexbox.

```
.container {
    display: flex;
    justify-content: center; /* Horizontal */
    align-items: center;     /* Vertical */
    height: 100vh;
}
```

```
div {
  transition: width 2s, height 2s, transform 2s;
}
div:hover {
  width: 300px;
  transform: rotate(180deg);
}
```

Practical Example: Hover Card Effect

Creating an interactive card that scales up:

```
.card {
  width: 250px;
  background: white;
  transition: transform 0.3s ease-in-out, box-shadow 0.3s;
}

.card:hover {
  transform: scale(1.05); /* Enlarge slightly */
  box-shadow: 0 10px 20px rgba(0,0,0,0.2);
}
```

3 Chapter 3: Bootstrap RWD

This document covers the principles of creating websites that look good on all devices and introduces the Bootstrap 5 framework for faster development.

Exam Tips & Hints

- **Viewport**: The `<meta name="viewport" ...>` tag is the single most common question about RWD setup.
- **Grid Sizing**: Understand the difference between `.container` (fixed widths at breakpoints) and `.container-fluid` (100% width always).
- **Breakpoints**: Memorize the standard Bootstrap breakpoints (sm, md, lg, xl, xxl).
- **12 Columns**: Every row in Bootstrap is divided into 12 columns. Ensure your `col-*` classes in a single row add up to 12 for standard layout.

3.1 Responsive Web Design (RWD) Fundamentals

Responsive Web Design is about creating websites that automatically adjust to different screen sizes, from mobile phones to large desktops.

3.1.1 The Viewport Meta Tag

To ensure proper rendering on mobile devices, always include the viewport meta tag in the `<head>`:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

3.1.2 Box-Sizing

Standard box model adds padding and borders to the width, which can break layouts. Use `box-sizing: border-box` to include them within the defined width:

```
* {
  box-sizing: border-box;
}
```

3.1.3 Responsive Images

Make images scale automatically to fit their container:

```
img {
  max-width: 100%;
  height: auto;
}
```

3.2 Layout Techniques

3.2.1 Grid System (12 Columns)

Modern responsive design often uses a 12-column grid. The page is divided into 12 equal parts, and elements are assigned a number of columns.

- **Float-based**: Older method using `float: left` and percentage widths.
- **Flexbox**: Modern method using `display: flex`.
- **CSS Grid**: Powerful layout engine using `display: grid`.

3.2.2 Media Queries (`@media`)

Allows applying CSS rules based on device characteristics (like width).

```
/* Styling for mobile (Mobile First) */
.column { width: 100%; }

/* Styling for desktop (larger than 768px) */
@media screen and (min-width: 768px) {
  .column { width: 50%; }
}
```

Practical Example: Mobile First Design

Standard workflow starting from mobile and scaling up:

```
/* Default (Mobile) */
.content {
  padding: 10px;
  font-size: 14px;
}

/* Tablets (min 600px) */
@media screen and (min-width: 600px) {
  .content { padding: 20px; }
}

/* Desktop (min 1024px) */
@media screen and (min-width: 1024px) {
  .content { font-size: 18px; }
}
```

3.3 Introduction to Bootstrap 5

Bootstrap is a free front-end framework for faster and easier web development. It includes design templates for typography, forms, buttons, tables, and many UI components.

3.3.1 Getting Started

Include Bootstrap via CDN in your `<head>`:

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
```

3.3.2 The Grid System

Built with flexbox and allows up to 12 columns.

- **Structure**: `.container` > `.row` > `.col-*`
- **Classes for breakpoints**:
  - `.col-`: Extra small (<576px)
  - `.col-sm-`: Small (≥576px)
  - `.col-md-`: Medium (≥768px)

- `.col-lg-`: Large (≥992px)
- `.col-xl-`: Extra Large (≥1200px)
- `.col-xxl-`: Extra Extra Large (≥1400px)

Practical Example: Bootstrap 12-Column Grid

A responsive row that changes layout based on screen size:

```
<div class="container">
  <div class="row">
    <!-- Full width on mobile, half on medium, third on large -->
    <div class="col-12 col-md-6 col-lg-4">Column 1</div>
    <div class="col-12 col-md-6 col-lg-4">Column 2</div>
    <div class="col-12 col-md-12 col-lg-4">Column 3</div>
  </div>
</div>
```

3.4 Bootstrap Components

Boostrap provides a wide range of pre-styled components:

- **Alerts**: Contextual feedback (e.g., `.alert-success`, `.alert-danger`).
- **Badges**: Labels and counts within other elements.
- **Buttons**: Custom styles for actions (e.g., `.btn`, `.btn-primary`, `.btn-outline-secondary`).
- **Cards**: Flexible content containers with headers, footers, and images.
- **Carousel**: A slideshow for cycling through images or text.
- **Navbar**: Responsive navigation headers.
- **Pagination**: Large hit areas for multi-page navigation.
- **Progress Bars**: Indicate work progress with labels.
- **Spinners & Toasts**: Loading indicators and push-notifications.

Practical Example: Bootstrap Components

Using a Navbar and a simple Alert:

```
<!-- Responsive Navbar -->
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <div class="container">
    <a class="navbar-brand" href="#">Logo</a>
    <button class="navbar-toggler" data-bs-toggle="collapse" data-bs-
        target="#myNav">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="myNav">
      <ul class="navbar-nav">
        <li class="nav-item"><a class="nav-link" href="#">Home</a></li>
      </ul>
    </div>
  </div>
</nav>

<!-- Success Alert -->
<div class="alert alert-success mt-3">
  Registration Successful!
</div>
```

Comprehensive Exam Prep: Bootstrap & RWD

Past Paper Questions

1. **MCQ:** Which Bootstrap class provides a responsive fixed-width container?
   a. .container
   b. .container-fluid
   c. .fixed-container
   d. .row

   Answer: a) .container (Note: .container-fluid is full width).

2. **MCQ:** How do you create a responsive grid system in Bootstrap?
   a. Using rows and columns
   b. Using flexbox
   c. Using tables
   d. Using divs

   Answer: a) Using rows and columns

Extra Practice Questions

Bootstrap Grid System

4. Which class indicates that an element should take up 6 columns on medium screens and larger?

   a. `.col-6-md`

   b. `.col-md-6`

   c. `.col-medium-6`

   d. `.col-xs-6`

   Answer: b) `.col-md-6`

5. What is the sum of columns in a single Bootstrap row?
   Answer: 12

Responsive Design Theory

Question: Write the standard Media Query break points for tablet and desktop.

```
/* Tablet (Assessment of around 768px) */
@media (min-width: 768px) { ... }

/* Laptop/Desktop (Assessment of around 992px or 1024px) */
@media (min-width: 992px) { ... }
```

Question: What does `<meta name="viewport" ...>` do?
Answer: It controls the page's dimensions and scaling, ensuring it renders correctly on mobile devices (e.g., setting width to device-width).

4 Chapter 4: JavaScript - The Language of the Web

JavaScript is a versatile, high-level language used for both client-side and server-side development. This chapter covers everything from basic syntax to advanced ES6 features and DOM manipulation.

Exam Tips & Hints

- **Variables**: Difference between `var`, `let`, and `const` is a classic theoretical question. Mention block-scope vs function-scope.
- **Hoisting**: Remember that `var` declarations are hoisted but not initialized. This can lead to `undefined` being logged if a variable is accessed before its declaration within the same scope.
- **Equality**: Understand the difference between `==` (loose, with coercion) and `===` (strict). Multiple MCQs often focus on this.
- **Array Methods**: `map()`, `filter()`, and `reduce()` are highly important. Be careful with `reverse()` and `sort()` as they modify the original array in-place.
- **Types**: Remember that `typeof []` is `"object"`.

4.1 JavaScript Fundamentals

4.1.1 Variables

- `var`: Function-scoped, can be redeclared and hoisted.
- `let`: Block-scoped, cannot be redeclared, not hoisted (preferred).
- `const`: Block-scoped, constant value (cannot be reassigned).

4.1.2 Data Types

- **Primitive**: `String`, `Number`, `Boolean`, `Undefined`, `Null`, `Symbol`, `BigInt`.
- **Complex**: `Object`, `Array`, `Function`.
- *Note:* `typeof null` returns `"object"` (a known bug).

4.1.3 Operators

- **Assignment**: `=`, `+=`, `-=`, etc.
- **Comparison**: `==` (equal value), `===` (equal value and type), `!=`, `!==`.
- **Logical**: `&&` (AND), `||` (OR), `!` (NOT).

4.1.4 Control Structures

- **Conditionals**: `if`, `else if`, `else`, `switch`.
- **Loops**:
  - `for`: Classic iterator.
  - `for...in`: Iterates over object keys.
  - `for...of`: Iterates over iterable values (like array elements).
  - `while` & `do...while`.

4.2 Strings and Arrays

4.2.1 String Methods

Common methods: `concat()`, `charAt()`, `replace()`, `toLowerCase()`, `toUpperCase()`, `trim()`, `split()`.

Ordering: `sort()` (alphabetical), `reverse()`.

Practical Example: Functional Programming

Using `map` and `filter` to process data:

```
const products = [
  { name: 'Laptop', price: 1200 },
  { name: 'Phone', price: 800 },
  { name: 'Tablet', price: 500 }
];

// 1. Filter products cheaper than 1000
const affordable = products.filter(p => p.price < 1000);

// 2. Map to an array of names
const names = affordable.map(p => p.name);

console.log(names); // ['Phone', 'Tablet']
```

4.3 Functions

4.3.1 Definitions

1. **Declaration**: `function name() { ... }` (Hoisted)
2. **Expression**: `const name = function() { ... }` (Anonymous or named)
3. **Arrow Function**: `const name = () => { ... }` (Shorter syntax, no `this` binding)

4.3.2 Parameters and Arguments

- **Default Parameters**: `function greet(name = "Guest") { ... }`
- **Arguments Object**: Array-like object containing all arguments passed to a function.
- **Rest Parameters**: `function sum(...nums) { ... }` (Collects remaining arguments into an array).

4.4 Object-Oriented JavaScript (Classes)

Classes are syntactical sugar over prototypes.

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // Getter
  get area() { return this.height * this.width; }
  // Setter
  set side(val) { this.height = val; this.width = val; }
}
const square = new Rectangle(10, 10);
```

Practical Example: Class Inheritance

Creating specialized classes from a base class:

```
class Animal {
  constructor(name) { this.name = name; }
  speak() { console.log(`${this.name} makes a noise.`); }
}

class Dog extends Animal {
  speak() { console.log(`${this.name} barks!`); }
}

const d = new Dog('Mitzie');
d.speak(); // Mitzie barks!
```

4.5 The Document Object Model (DOM)

The DOM is a programming interface for HTML documents. It represents the page as a tree of objects.

4.5.1 Selecting Elements

- `document.getElementById('id')`
- `document.querySelector('.selector')`
- `document.querySelectorAll('.selector')`

4.5.2 Manipulating Elements

- **Content**: `element.innerHTML`, `element.textContent`, `element.value` (for inputs).
- **Attributes**: `getAttribute()`, `setAttribute()`, `removeAttribute()`, `hasAttribute()`.
- **Styles**: `element.style.color = 'red';` (uses camelCase for CSS properties).
- **Creation**: `document.createElement('div')`, `parent.appendChild(child)`.

Comprehensive Exam Prep: JavaScript

Past Paper Questions

1. **MCQ:** What will `console.log(typeof [])` output? **Answer:** object (Arrays are objects in JS).

2. **Code Analysis (Sessional 2 2024):**

```
var arr1 = "web".split('');
var arr2 = arr1.reverse();
var arr3 = "Programming".split('');
arr2.push(arr3);
console.log("array 2: " + arr2);
```

   Explanation: 1. `arr1` = ['w','e','b']. 2. `arr2` reference refers to `arr1` reversed: ['b','e','w']. Note: `reverse()` mutates in place if referencing same array, but here `split` creates new. 3. `arr3` = ['P','r',...]. 4. `push` adds the *entire array* `arr3` as a single element at the end.
   Output: `b,e,w,P,r,o,g,r,a,m,m,i,n,g` (Browsers flatten for display string).

3. **DOM Manipulation:** Write code to display user name N times.

```
function displayName() {
  const num = document.getElementById('num').value;
  const list = document.getElementById('list');
  list.innerHTML = ''; // Clear previous
  for(let i=0; i<num; i++) {
     const li = document.createElement('li');
     li.textContent = "My Name";
     list.appendChild(li);
  }
}
```

Extra Practice Questions

ES6 Features

4. Which keyword allows declaring a variable that cannot be reassigned? **Answer:** `const`

5. Rewrite this using Arrow Functions:

```
    function sum(a, b) { return a + b; }
```

   Answer: `const sum = (a, b) => a + b;`

Advanced Scoping

Question: Output of the following?

```
console.log(a);
var a = 5;
```

Answer: `undefined`. (Reason: `var` declarations are hoisted, but initializations are not).

```
const btn = document.querySelector('#myBtn');
btn.addEventListener('click', (event) => {
  console.log('Button clicked!', event.target);
});
```

Practical Example: Dynamic DOM Update

Changing text and style on a button click:

```
const heading = document.querySelector('h1');
const colorBtn = document.querySelector('#colorBtn');

colorBtn.addEventListener('click', () => {
  heading.textContent = 'Color Changed!';
  heading.style.color = 'blue';

  // Create a new element
  const p = document.createElement('p');
  p.innerText = 'This was added dynamically.';
  document.body.appendChild(p);
});
```

Comprehensive Exam Prep: JavaScript

Past Paper Questions

1. **MCQ:** What will `console.log(typeof [])` output? **Answer:** object (Arrays are objects in JS).

2. **Code Analysis (Sessional 2 2024):**

```
var arr1 = "web".split('');
var arr2 = arr1.reverse();
var arr3 = "Programming".split('');
arr2.push(arr3);
console.log("array 2: " + arr2);
```

   Explanation: 1. `arr1` = ['w','e','b']. 2. `arr2` reference refers to `arr1` reversed: ['b','e','w']. Note: `reverse()` mutates in place if referencing same array, but here `split` creates new. 3. `arr3` = ['P','r',...]. 4. `push` adds the *entire array* `arr3` as a single element at the end.
   Output: `b,e,w,P,r,o,g,r,a,m,m,i,n,g` (Browsers flatten for display string).

3. **DOM Manipulation:** Write code to display user name N times.

```
function displayName() {
  const num = document.getElementById('num').value;
  const list = document.getElementById('list');
  list.innerHTML = ''; // Clear previous
  for(let i=0; i<num; i++) {
     const li = document.createElement('li');
     li.textContent = "My Name";
     list.appendChild(li);
  }
}
```

Extra Practice Questions

ES6 Features

4. Which keyword allows declaring a variable that cannot be reassigned? **Answer:** `const`
5. Rewrite this using Arrow Functions:

```
function sum(a, b) { return a + b; }
```

Answer: `const sum = (a, b) => a + b;`

Advanced Scoping

Question: Output of the following?

```
console.log(a);
var a = 5;
```

Answer: `undefined`. (Reason: `var` declarations are hoisted, but initializations are not).

5 Chapter 5: React - Modern Front-end Development

React is a powerful JavaScript library for building user interfaces, developed by Facebook. It focuses on reusable components and efficient rendering via the Virtual DOM.

Exam Tips & Hints

- **Virtual DOM**: Understand that the Virtual DOM is a lightweight copy of the Real DOM. React uses it to calculate the minimum changes (Reconciliation) before updating the actual browser DOM.
- **Hooks**: `useState` and `useEffect` are extremely common in practical coding questions.
- **Class Context**: Remember that in class components, custom methods must be bound to `this` or written as arrow functions to access state.
- **Data Flow**: Data flows *down* via props and *up* via function callbacks.
- **Keys**: Expect a question on why `key` props are needed when mapping lists (helping React's reconciliation process).

5.1 Core Concepts

5.1.1 React vs React Native

- **React**: For web applications (Single Page Apps).
- **React Native**: For mobile application development (Android/iOS).

5.1.2 The Virtual DOM

React creates an in-memory database cache (Virtual DOM) that tracks changes. When the UI changes, React compares the Virtual DOM to the real DOM and updates only the necessary parts, leading to better performance.

5.1.3 JSX (JavaScript XML)

A syntax extension that looks like HTML but lives in JavaScript.

- Must have a single root element.
- Use `className` instead of `class`.
- Use `htmlFor` instead of `for` for labels.

Class Components (Container): Use ES6 classes and have access to `this`, `state`, and lifecycle methods.

Practical Example: Simple Functional Component

A reusable component that displays a greeting:

```
import React from 'react';

const Welcome = (props) => {
  return (
    <div className="welcome-card">
      <h1>Hello, {props.name}!</h1>
      <p>Welcome to our React application.</p>
    </div>
  );
};

export default Welcome;
```

5.2 State and Props

5.2.1 State

An internal data store for a component. When state changes, the component re-renders.

- Always use `this.setState()` in class components to update state.
- In functional components, use the `useState` hook.

5.2.2 Props (Properties)

Data passed from a parent component to a child component. Props are read-only for the child.

Practical Example: State and Props

Managing a counter with `useState`:

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

5.3 Handling Events

- React events are named using camelCase (e.g., `onClick`, `onSubmit`).
- **`this` Context**: In class components, `this` is lost in custom functions. Solutions:
  - Bind in constructor: `this.myFunc = this.myFunc.bind(this)`.
  - Use **Arrow Functions**: `myFunc = () => { ... }`.

5.4 Forms and Interactivity

- **Controlled Components**: Input values are driven by state.
- `e.preventDefault()`: Used in form submission to stop the default page refresh.
- **Functions as Props**: Passing a function from parent to child allows the child to communicate back to the parent (e.g., deleting an item from a list held in parent state).

5.5 Development Tools and Setup

- **React Dev Tools**: Browser extension for inspecting component hierarchies and state/props.
- **Create React App (CRA)**: A standard toolchain for setting up a modern React project.
  - `npx create-react-app my-app`
  - `npm start`: Runs the app in development mode.

5.6 React Router

Used to handle navigation in a Single Page App without refreshing the page.

- `<BrowserRouter>`: Wraps the whole app.
- `<Route>`: Defines a path and the component to render.
- `<Link>` and `<NavLink>`: Replace `<a>` tags for internal navigation.
- **Route Parameters**: `path="/post/:id"`.

5.7 Advanced Topics

5.7.1 Higher-Order Components (HOCs)

A function that takes a component and returns a new ("enhanced") component. Used for shared logic like authentication or loading states.

```
const Protected = withAuth(Dashboard);
```

Comprehensive Exam Prep: React

Past Paper Questions (Sessional 2 2024 & Final)

1. **Theory:** Difference between Real DOM and Virtual DOM?
   - **Real DOM**: The actual HTML structure. Slow updates (heavy reflow/repaint).
   - **Virtual DOM**: Lightweight JavaScript object copy. React updates this first, compares (diffing), and patches only changes to Real DOM (reconciliation).
2. **Theory:** List common React events.
   Answer: `onClick`, `onChange`, `onSubmit`, `onMouseEnter`.
3. **Theory:** What are Keys? Are they necessary?
   Answer: Keys help React identify which items in a list have changed, added, or removed. They are necessary for performance and avoiding bugs in dynamic lists.
4. **Task:** Create a Counter Component (Class-based).

```
class Counter extends React.Component {
  state = { count: 0 };
  increment = () => this.setState({ count: this.state.count + 1 });
  render() {
    return <button onClick={this.increment}>{this.state.count}</
       button>;
  }
}
```

Extra Practice Questions

Hooks Functional Components

1. How do you replicate `componentDidMount` with `useEffect`?
   Answer: `useEffect(() => { ... }, [])` (Empty dependency array).
2. **Code Snippet:** Create a component using `useState` to toggle text visibility.

```
function Toggle() {
  const [show, setShow] = useState(true);
  return (
    <div>
      <button onClick={() => setShow(!show)}>Toggle</button>
      {show && <p>Hidden Text</p>}
    </div>
  );
}
```

5.7.2 Hooks (ES6+)

- `useState`: Adds state to functional components.
- `useEffect`: Handles side effects (like data fetching). Replaces lifecycle methods like `componentDidMount`.

5.7.3 Data Fetching (Axios)

Axios is a popular library to fetch data from APIs.

```
useEffect(() => {
```

```
  axios.get('https://api.example.com/posts')
    .then(res => setPosts(res.data));
}, []);
```

Practical Example: Data Fetching Hook

Fetching user profile on mount:

```
import { useState, useEffect } from 'react';
import axios from 'axios';

function UserProfile({ userId }) {
  const [user, setUser] = useState(null);

  useEffect(() => {
    // Side effect: fetch data
    axios.get(`/api/users/${userId}`)
      .then(response => setUser(response.data))
      .catch(error => console.error(error));
  }, [userId]); // Runs when userId changes

  if (!user) return <div>Loading...</div>;
  return <h1>{user.name}</h1>;
}
```

6 Chapter 6: Express.js - Web Application Framework

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is "unopinionated," meaning it gives developers freedom in how they structure their apps.

Exam Tips & Hints

- **Middleware**: The `next()` function is critical. If you don't call it, the request hangs.
- **Routing**: Route parameters (e.g., `:id`) vs query strings (`?id=1`).
- **Methods**: Use `GET` for retrieval and `POST` or `PUT` for data submission/update.
- **RegEx Routing**: Practice the special characters (`?`, `+`, `*`) in route paths.

6.1 Getting Started

6.1.1 Setup

1. Create a directory: `mkdir myapp`
2. Initialize npm: `npm init`
3. Install Express: `npm install express`

6.1.2 Hello World Example

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

6.2 Modules and Asynchrony

- **Modules**: Use `module.exports` to share code across files and `require()` to import it.
- **Async Programming**: Node is single-threaded and non-blocking. Use asynchronous APIs (callbacks, Promises) to avoid blocking the event loop.
- **Error-First Callbacks**: A convention where the first argument of a callback is the error object.

6.3 Middleware

Middleware functions are the backbone of Express. They have access to the request (`req`), response (`res`), and the `next` function in the application's request-response cycle.

Third-party: Community modules like `morgan` (logging) or `cors`.

Practical Example: Custom Logger Middleware

A function that logs the request method and URL:

```
const logger = (req, res, next) => {
  console.log(`${new Date().toISOString()} - ${req.method} ${req.url}`)
      ;
  next(); // Pass control to the next handler
};

app.use(logger); // Apply to all routes
```

Chaining: Use `app.route()` to define multiple methods for a single path.

Practical Example: URL Parameters

Extracting data from the request URL:

```
// GET /shop/electronics/iphone
app.get('/shop/:category/:item', (req, res) => {
  const { category, item } = req.params;
  res.send(`Searching for ${item} in ${category}`);
});
```

6.4 Templating with EJS

EJS (Embedded JavaScript) is a template engine that lets you generate HTML with plain JavaScript.

- **Setup**: `app.set('view engine', 'ejs');`
- **Syntax**:
  - `<% %>`: Executes logic (loops, conditionals).
  - `<%= %>`: Outputs a value to the page.
- **Rendering**: `res.render('index', { name: 'John' });`

Practical Example: RESTful API Implementation

Standard CRUD endpoints for a "Task" resource:

```
let tasks = [{ id: 1, title: 'Learn Express' }];

app.get('/tasks', (req, res) => res.json(tasks));

app.post('/tasks', (req, res) => {
  const newTask = { id: Date.now(), ...req.body };
  tasks.push(newTask);
  res.status(201).json(newTask);
});

app.delete('/tasks/:id', (req, res) => {
  tasks = tasks.filter(t => t.id != req.params.id);
  res.status(204).send();
});
```

Comprehensive Exam Prep: Express.js

Past Paper Questions

1. **MCQ:** Which method is used to define a GET route in Express? **Answer:** `app.get()`
2. **MCQ:** Which middleware function is used to parse incoming request bodies? **Answer:** `body-parser` (or built-in `express.json()` in modern versions).
3. **Route Logic:** What is the output of the following sequence?

```
app.use((req, res, next) => { console.log('A'); next(); });
app.get('/test', (req, res) => { console.log('B'); });
```

Answer: Output will be `A` then `B`. The middleware runs for all routes, then `next()` passes control to the specific GET handler.

Extra Practice Questions

Middleware & Routing

Question: Write a custom middleware that logs the Request URL and Method.

```
app.use((req, res, next) => {
    console.log(`Method: ${req.method}, URL: ${req.url}`);
    next(); // Critical to prevent hanging
});
```

Question: How do you access the query parameter `?id=123` in Express?
Answer: `req.query.id`
Question: How do you access the URL parameter `/users/:uId` (where uId is 123)?
Answer: `req.params.uId`

7 Chapter 7: MongoDB and Mongoose - Database Management

MongoDB is a NoSQL, document-oriented database that stores data in flexible, JSON-like documents. This means fields can vary from document to document and data structure can be changed over time.

Exam Tips & Hints

- **RDBMS vs NoSQL**: Be prepared to compare Tables/Collections and Rows/Documents.
- **Mongoose Schema**: Know how to define types and required fields in a Mongoose schema.
- **Relationships**: Understand the pros and cons of Embedding vs Referencing.
- **`_id`**: Every document has a unique `_id` field by default.

7.1 Core Concepts

RDBMS Terminology	MongoDB Terminology
Database	Database
Table	Collection
Row / Tuple	Document
Column	Field
Table Join	Embedded Documents / $lookup
Primary Key	Primary Key (Default `_id`)

7.1.1 Advantages of MongoDB

- **Schema-less**: Documents in a collection don't need the same fields.
- **Easy Scalability**: Designed to scale out across servers.
- **High Performance**: Uses internal memory for faster data access.

7.2 MongoDB Shell Commands

- `use mydb`: Switch to or create a database.
- `db.createCollection('users')`: Create a new collection.
- `db.users.insert({name: 'John'})`: Insert a document.
- `db.users.find()`: Query all documents.
- `db.users.update({name: 'John'}, {$set: {age: 30}})`: Update a document.
- `db.users.remove({name: 'John'})`: Remove a document.
- `db.dropDatabase()`: Delete the current database.

7.3 Using MongoDB with Node.js

7.3.1 Native MongoDB Driver

Requires the `mongodb` npm package.

```
const { MongoClient } = require('mongodb');
const url = "mongodb://localhost:27017/";

MongoClient.connect(url, (err, db) => {
  if (err) throw err;
  const dbo = db.db("mydb");
  dbo.collection("customers").findOne({}, (err, result) => {
    console.log(result.name);
    db.close();
  });
});
```

7.4 Mongoose - Elegant Object Modeling

Mongoose provides a schema-based solution to model your application data.

Schema: Defines the structure of the document.

Model: A constructor that creates documents based on the schema.

7.4.1 Defining a Schema and Model

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: String,
  email: { type: String, required: true },
  createdAt: { type: Date, default: Date.now }
});

const User = mongoose.model('User', userSchema);
```

Practical Example: Schema and Validation

Defining a strict schema for a "Product" collection:

```
const productSchema = new mongoose.Schema({
  name: { type: String, required: [true, 'Name is required'] },
  price: { type: Number, min: 0 },
  category: { type: String, enum: ['Electronics', 'Books', 'Clothing']
     },
  inStock: { type: Boolean, default: true }
});

const Product = mongoose.model('Product', productSchema);
```

Many-to-Many: Create a third "junction" collection to track relationships between two other collections.

Practical Example: Mongoose Relationships

Using `ref` and `populate()` to link data:

```
// Post Schema
const postSchema = new mongoose.Schema({
  title: String,
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }
});

// Query with Populate
Post.find()
  .populate('author') // Replaces ID with actual User object
  .then(posts => console.log(posts));
```

Comprehensive Exam Prep: MongoDB Mongoose

Past Paper Questions

1. **MCQ:** Which method is used to retrieve all documents from a MongoDB collection?
   a. find()
   b. getAll()
   c. retrieve()
   d. fetch()

   Answer: a) `find()`

2. **MCQ:** What does CRUD stand for? **Answer:** Create, Read, Update, Delete.

3. **Code Task:** Write code to connect to MongoDB using Mongoose.

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/mydb', {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log("Connected"));
```

Extra Practice Questions

Schemas and Models

Question: Define a Mongoose Schema for a 'Book' with `title` (String) and `author` (String).

```
const bookSchema = new mongoose.Schema({
    title: { type: String, required: true },
    author: String
});
const Book = mongoose.model('Book', bookSchema);
```

Question: How do you find a single document where `name` is "Alice"?
Answer: `User.findOne({ name: 'Alice' }).then(...)`

8 Chapter 8: Node.js - Server-Side JavaScript Runtime

Node.js is a powerful JavaScript platform built on Chrome's V8 engine. It allows developers to use JavaScript to write server-side code, enabling "Full-stack JavaScript" development.

Exam Tips & Hints

- **Event Loop**: A fundamental concept. Understand the roles of the Call Stack, libuv, and the Task Queue.
- **NPM**: Know the difference between `dependencies` and `devDependencies`.
- **Global Objects**: Remember `__dirname` and `__filename` are provided by the Node.js wrapper, not part of standard JS.
- **Callback Pattern**: Error-first callbacks (`err` as the first argument) is a standard convention.

8.1 Core Principles

- **Asynchronous & Event-Driven**: All APIs of Node.js library are asynchronous (non-blocking). A Node-based server never waits for an API to return data.
- **Single-Threaded**: Node.js uses a single-threaded model with event looping. This makes it highly scalable for I/O intensive apps, though it's not ideal for CPU-intensive tasks.
- **Platform Independent**: Runs on Windows, Linux, and macOS.

8.2 The Node.js Event Loop

- **libuv**: A library that Node.js uses to perform asynchronous I/O and manage a thread pool.
- **Call Stack**: Where the code being executed lives.
- **Event Queue**: When an async task (like a timer or file read) completes, its callback is sent to this queue.
- **Event Loop**: Monitors the call stack and event queue. If the stack is empty, it pushes the first task from the queue to the stack for execution.

8.3 Node Package Manager (NPM)

NPM is the default package manager for Node.js.

- `package.json`: Holds metadata about the project and its dependencies.
- `package-lock.json`: Records the exact version of every installed dependency to ensure consistent builds across environments.
- **Commands**:
  - `npm init`: Initialize a new project.
  - `npm install <pkg>`: Install a package and add it to `dependencies`.
  - `npm ci`: A cleaner, faster install for automated environments (CI/CD).
  - `npm audit`: Checks for security vulnerabilities in dependencies.

8.4 Global Objects and Modules

- `__dirname`: Directory name of the current script.
- `__filename`: File name of the current script.
- `process`: Provides information about, and control over, the current Node.js process.
- `console`: Used for printing to stdout and stderr.

Net: Used to create TCP servers and clients.

Practical Example: Reading and Writing Files

Using the built-in `fs` module (promises version):

```
const fs = require('fs').promises;

async function manageFiles() {
  try {
    // Write to a file
    await fs.writeFile('test.txt', 'Hello Node.js!');
    // Read from the file
    const data = await fs.readFile('test.txt', 'utf8');
    console.log(data);
  } catch (err) {
    console.error(err);
  }
}
manageFiles();
```

Practical Example: Simple HTTP Server

Creating a server without any frameworks:

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello from Node.js Server!');
});

server.listen(3000, '127.0.0.1', () => {
  console.log('Server running at http://127.0.0.1:3000/');
});
```

Comprehensive Exam Prep: Node.js (Advanced)

Past Paper Questions

1. **MCQ:** What is the default port for a Node.js application (conventionally)? **Answer:** 3000 (though it can be anything distinct).
2. **MCQ:** Which command is used to initialize a new Node.js project? **Answer:** `npm init`
3. **MCQ:** Which of the following is not a valid JS data type? **Answer:** Character (JS has String, no distinct Char type).

Extra Practice Questions

Architecture Globals

Question: Explain the Node.js Event Loop.
Answer: Node.js is single-threaded. the Event Loop handles asynchronous callbacks. It offloads operations (like I/O) to the system kernel (libuv) and puts their callbacks into a queue. When the Call Stack is empty, the Event Loop pushes tasks from the queue to the stack.
Question: What is the difference between `module.exports` and `exports`?
Answer: `exports` is a reference to `module.exports`. If you assign a new object to `exports`, you break the link. Always use `module.exports` when exporting a single class or function.
Question: What does `__dirname` return?
Answer: The absolute path of the directory containing the currently executing file.

9 Chapter 9: Redux - State Management

Redux is a predictable state container for JavaScript apps. It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.

Exam Tips & Hints

- **Principles**: Single source of truth (Store), State is read-only (Actions), Changes are made with pure functions (Reducers).
- **Data Flow**: Unidirectional data flow: Action → Reducer → Store → View.
- **Hooks**: Familiarize yourself with `useSelector` and `useDispatch` for modern Redux.
- **Immutability**: Never mutate the state directly in a reducer; always return a new object/array.

9.1 Core Concepts

1. **Store**: The centralized place where the state of the application is stored.
2. **Actions**: Plain JavaScript objects that describe "what happened". Every action must have a `type` property.
3. **Reducers**: Pure functions that specify how the application's state changes in response to an action.
4. **Dispatch**: The method used to send actions to the store.
5. **Connect**: A function (mostly used in class components) from `react-redux` to link React components to the Redux store.

Practical Example: Basic Redux Setup

A simple counter example:

```
// 1. Action Types
const INCREMENT = 'INCREMENT';

// 2. Action Creator
const incrementAction = () => ({ type: INCREMENT });

// 3. Reducer
const counterReducer = (state = { count: 0 }, action) => {
  switch (action.type) {
    case INCREMENT:
      return { ...state, count: state.count + 1 };
    default:
      return state;
  }
};

// 4. Store
import { createStore } from 'redux';
const store = createStore(counterReducer);

// 5. Usage
store.dispatch(incrementAction());
console.log(store.getState()); // { count: 1 }
```

Comprehensive Exam Prep: Redux

Past Paper Questions (Final 2024)

1. **Theory:** Explain the Redux concept using practical example code.
   - **Store**: Holds the state tree.
   - **Action**: Dispatched to trigger changes (must have `type`).
   - **Reducer**: Pure function `(state, action) => newState`.

   (Refer to Chapter 9 notes for the full Book List code example)

Extra Practice Questions

Core Concepts

Question: What is the "Single Source of Truth"?
Answer: In Redux, the state of your whole application is stored in an object tree within a single store.
Question: Why must Reducers be "pure functions"?
Answer: They must not mutate the existing state. They should take the previous state and an action, and return a *new* state object. This makes state changes predictable and allows features like time-travel debugging.
Task: Write a reducer for a "Counter" that handles `INCREMENT`.

```
const initialState = { count: 0 };
function counterReducer(state = initialState, action) {
  switch(action.type) {
    case 'INCREMENT':
      return { ...state, count: state.count + 1 };
    default:
      return state;
  }
}
```

10 Extra MCQs

No extra MCQs available.