Web Development Cheatsheet

Essential operations for building modern web applications

This cheatsheet provides a quick reference to fundamental web development concepts, syntax, and best practices, ideal for beginners starting their journey in creating interactive and responsive websites.

HTML Structure
Build semantic document foundations

CSS Styling
Design beautiful and responsive layouts

JavaScript Logic
Add interactivity and dynamic behavior

DOM Manipulation
Control and update page content

Responsive Design
Create mobile-friendly experiences

HTML Fundamentals & Document Structure

Basic HTML Structure: `<!DOCTYPE>`

Create the foundation of every web page.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Web Page</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Hello World</h1>
  <script src="script.js"></script>
</body>
</html>
```

Semantic Elements: `<header>` / `<main>` / `<footer>`

Use meaningful HTML semantic elements for better structure.

```
<header>
  <nav>
    <ul>
      <li><a href="#home">Home</a></li>
      <li><a href="#about">About</a></li>
    </ul>
  </nav>
</header>
<main>
  <section>
    <h1>Welcome</h1>
    <p>Main content here</p>
  </section>
</main>
<footer>
  <p>© 2024 My Website</p>
</footer>
```

Text Elements: `<h1>` to `<h6>` / `<p>`

Structure content with proper heading hierarchy and paragraphs.

```
<h1>Main Title</h1>
<h2>Section Heading</h2>
<h3>Subsection</h3>
<p>This is a paragraph with <strong>bold text</strong> and <em>italic text</em>.</p>
<p>Another paragraph with a <a href="https://example.com">link</a>.</p>
```

Lists: `<ul>` / `<ol>` / `<li>`

Create organized lists of information.

```
<!-- Unordered list -->
<ul>
  <li>First item</li>
  <li>Second item</li>
  <li>Third item</li>
</ul>
<!-- Ordered list -->
<ol>
  <li>Step 1</li>
  <li>Step 2</li>
  <li>Step 3</li>
</ol>
```

Images & Media: `<img>` / `<video>` / `<audio>`

Embed multimedia content with proper attributes.

```
<!-- Image with alt text -->
<img src="image.jpg" alt="Description of image" width="500">
<!-- Video element -->
<video controls width="400">
  <source src="video.mp4" type="video/mp4">
  Your browser doesn't support video.
</video>
<!-- Audio element -->
<audio controls>
  <source src="audio.mp3" type="audio/mpeg">
</audio>
```

Tables: `<table>` / `<tr>` / `<td>`

Display tabular data with proper structure.

```
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
      <th>City</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>John</td>
      <td>25</td>
      <td>New York</td>
    </tr>
  </tbody>
</table>
```

Forms & User Input

Create interactive forms to collect user data and handle various input types.

01

Form Structure: `<form>`

Create the container for user inputs and controls.

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

  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>

  <button type="submit">Submit</button>
</form>
```

02

Input Types: `type="text"` / `type="email"`

Use appropriate input types for different data.

```
<input type="text" placeholder="Enter your name">
<input type="email" placeholder="email@example.com">
<input type="password" placeholder="Password">
<input type="number" min="1" max="100">
<input type="date">
<input type="checkbox" id="agree">
<input type="radio" name="gender" value="male">
<input type="file" accept=".jpg,.png">
```

03

Form Controls: `<select>` / `<textarea>`

Provide various ways for users to input information.

```
<select name="country" id="country">
  <option value="">Select a country</option>
  <option value="us">United States</option>
  <option value="ca">Canada</option>
</select>

<textarea name="message" rows="4" cols="50"
          placeholder="Enter your message"></textarea>
```

CSS Fundamentals & Styling

CSS Selectors: `element` / `.class` / `#id`

Target HTML elements for styling with different selector types.

```
/* Element selector */
h1 {
  color: blue;
  font-size: 2rem;
}
/* Class selector */
.highlight {
  background-color: yellow;
  padding: 10px;
}
/* ID selector */
#header {
  background-color: navy;
  color: white;
}
/* Descendant selector */
.container p {
  line-height: 1.6;
}
```

Box Model: `margin` / `padding` / `border`

Control spacing and layout with the CSS box model.

```
.box {
  width: 300px;
  height: 200px;
  margin: 20px;          /* Outside spacing */
  padding: 15px;         /* Inside spacing */
  border: 2px solid black; /* Border properties */
}
/* Shorthand properties */
.element {
  margin: 10px 20px;     /* top/bottom left/right */
  padding: 10px 15px 20px 25px; /* top right bottom left */
  border-radius: 5px;    /* Rounded corners */
}
```

Flexbox: `display: flex`

Create flexible and responsive layouts easily.

```
.container {
  display: flex;
  justify-content: center;  /* Horizontal alignment */
  align-items: center;      /* Vertical alignment */
  gap: 20px;                /* Space between items */
}
.flex-item {
  flex: 1;                  /* Equal width items */
}
/* Flexbox direction */
.column-layout {
  display: flex;
  flex-direction: column;
}
```

Grid Layout: `display: grid`

Create complex two-dimensional layouts.

```
.grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr); /* 3 equal columns */
  grid-gap: 20px;
  grid-template-areas:
    "header header header"
    "sidebar main main"
    "footer footer footer";
}
.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.main { grid-area: main; }
```

JavaScript Basics & Programming Fundamentals

Variables: `let` / `const` / `var`

Store and manipulate data with different variable declarations.

```
// Modern variable declarations
let name = "John";        // Can be reassigned
const age = 25;           // Cannot be reassigned
const colors = ["red", "blue"]; // Array (contents can change)

// Variable types
let message = "Hello World";  // String
let count = 42;               // Number
let isActive = true;          // Boolean
let data = null;              // Null
let user = {                  // Object
  name: "Alice",
  email: "alice@example.com"
};
```

Functions: `function` / Arrow Functions

Create reusable blocks of code with different function syntax.

```
// Function declaration
function greet(name) {
  return `Hello, ${name}!`;
}
// Arrow function
const add = (a, b) => a + b;
// Arrow function with block
const calculateArea = (width, height) => {
  const area = width * height;
  return area;
};
// Function with default parameters
function createUser(name, age = 18) {
  return { name, age };
}
```

Conditional Logic: `if` / `else` / `switch`

Control program flow with conditional statements.

```
// If/else statement
if (age >= 18) {
  console.log("Adult");
} else if (age >= 13) {
  console.log("Teenager");
} else {
  console.log("Child");
}
// Ternary operator
const status = age >= 18 ? "adult" : "minor";
// Switch statement
switch (day) {
  case "Monday":
    console.log("Start of work week");
    break;
  case "Friday":
    console.log("TGIF!");
    break;
  default:
    console.log("Regular day");
}
```

Loops: `for` / `while` / Array Methods

Iterate through data and repeat operations.

```
// For loop
for (let i = 0; i < 5; i++) {
  console.log(i);
}
// For...of loop
for (const item of items) {
  console.log(item);
}
// Array methods
const numbers = [1, 2, 3, 4, 5];
numbers.forEach(num => console.log(num));
const doubled = numbers.map(num => num * 2);
const evens = numbers.filter(num => num % 2 === 0);
const sum = numbers.reduce((total, num) => total + num, 0);
```

DOM Manipulation & Events

Selecting Elements: `querySelector` / `getElementById`

Find and access HTML elements in JavaScript.

```
// Select single elements
const title = document.getElementById("title");
const button = document.querySelector(".btn");
const firstParagraph = document.querySelector("p");
// Select multiple elements
const allButtons = document.querySelectorAll("button");
const allParagraphs = document.getElementsByTagName("p");
// Check if element exists
if (button) {
  button.style.color = "blue";
}
```

Event Handling: `addEventListener`

Respond to user interactions and browser events.

```
// Click event
button.addEventListener("click", function() {
  alert("Button clicked!");
});
// Form submit event
form.addEventListener("submit", function(e) {
  e.preventDefault(); // Prevent form submission
  const formData = new FormData(form);
  console.log(formData.get("username"));
});
// Keyboard events
document.addEventListener("keydown", function(e) {
  if (e.key === "Enter") {
    console.log("Enter key pressed");
  }
});
```

Modifying Content: `innerHTML` / `textContent`

Change the content and attributes of HTML elements.

```
// Change text content
title.textContent = "New Title";
title.innerHTML = "<strong>Bold Title</strong>";
// Modify attributes
button.setAttribute("disabled", "true");
const src = image.getAttribute("src");
// Add/remove classes
button.classList.add("active");
button.classList.remove("hidden");
button.classList.toggle("highlighted");
```

Creating Elements: `createElement` / `appendChild`

Dynamically create and add new HTML elements.

```
// Create new element
const newDiv = document.createElement("div");
newDiv.textContent = "New content";
newDiv.className = "highlight";
// Add to page
document.body.appendChild(newDiv);
// Create list item
const li = document.createElement("li");
li.innerHTML = "<a href='#'>New Link</a>";
document.querySelector("ul").appendChild(li);
// Remove element
const oldElement = document.querySelector(".remove-me");
oldElement.remove();
```

Responsive Design & CSS Media Queries

Create websites that work perfectly on all devices and screen sizes.

Viewport Meta Tag: `viewport`

Set up proper viewport for responsive design.

```
<meta name="viewport"
      content="width=device-width, initial-scale=1.0">
/* CSS for responsive images */
img {
  max-width: 100%;
  height: auto;
}
/* Responsive container */
.container {
  width: 100%;
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 20px;
}
```

Flexible Units: `rem` / `em` / `%` / `vw` / `vh`

Use relative units for scalable and responsive designs.

```
/* Relative to root font-size */
h1 { font-size: 2rem; }    /* 32px if root is 16px */
/* Relative to parent font-size */
p { font-size: 1.2em; }    /* 1.2 times parent size */
/* Percentage based */
.sidebar { width: 30%; }   /* 30% of parent width */
/* Viewport units */
.hero {
  height: 100vh;           /* Full viewport height */
  width: 100vw;            /* Full viewport width */
}
```

Media Queries: `@media`

Apply different styles based on screen size and device characteristics.

```
/* Mobile first approach */
.grid {
  display: grid;
  grid-template-columns: 1fr; /* Single column on mobile */
  gap: 20px;
}
/* Tablet and up */
@media (min-width: 768px) {
  .grid {
    grid-template-columns: repeat(2, 1fr); /* 2 columns */
  }
}
/* Desktop and up */
@media (min-width: 1024px) {
  .grid {
    grid-template-columns: repeat(3, 1fr); /* 3 columns */
  }
}
```

Responsive Typography: `clamp()`

Create fluid typography that scales with screen size.

```
/* Fluid typography */
h1 {
  font-size: clamp(1.5rem, 4vw, 3rem);
  /* Min: 1.5rem, Preferred: 4vw, Max: 3rem */
}
/* Responsive spacing */
section {
  padding: clamp(2rem, 5vw, 6rem) clamp(1rem, 3vw, 3rem);
}
/* Container queries (newer browsers) */
@container (min-width: 400px) {
  .card {
    display: flex;
  }
}
```

Debugging & Browser Developer Tools

Console Methods: `console.log()` / `console.error()`

Debug and monitor your code with console output.

```
// Basic logging
console.log("Hello, world!");
console.log("User data:", userData);
// Different log levels
console.info("Information message");
console.warn("Warning message");
console.error("Error message");
// Grouping logs
console.group("User Details");
console.log("Name:", user.name);
console.log("Email:", user.email);
console.groupEnd();
```

Debugging Techniques: `debugger` / Breakpoints

Pause code execution to inspect variables and program state.

```
function calculateTotal(items) {
  let total = 0;
  debugger; // Code will pause here when dev tools open

  for (let item of items) {
    total += item.price;
    console.log("Current total:", total);
  }
  return total;
}
// Error handling
try {
  const result = riskyFunction();
} catch (error) {
  console.error("Error occurred:", error.message);
}
```

Browser DevTools: Elements / Console / Network

Use browser tools to inspect HTML, debug JavaScript, and monitor network requests.

```
// Inspect elements in console
$0 // Currently selected element in Elements tab
$1 // Previously selected element
// Query elements from console
$("selector")  // Same as document.querySelector
$$("selector") // Same as document.querySelectorAll
// Monitor functions
monitor(functionName) // Log when function is called
// Performance timing
console.time("operation");
// ... some code ...
console.timeEnd("operation");
```

Error Types: `TypeError` / `ReferenceError`

Understand common JavaScript errors and how to fix them.

```
// Common errors and solutions
// ReferenceError: Variable not defined
// console.log(undefinedVariable); //
```