CODER STAR

HTML Cheat Sheet

CSS cheat Sheet

JavaScript Cheat Sheet

HTML Cheat Sheet

HTML Document Structure:

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
</head>
<body>
  <!-- Content goes here -->
</body>
</html>
```

Basic HTML Elements:

```
<!-- Headings -->
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
<!-- Paragraph -->
```

```
<p>This is a paragraph.</p>
<!-- Links -->
<a href="https://example.com">Link Text</a>
<!-- Images -->
<img src="image.jpg" alt="Image Description">
<!-- Lists -->
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>

<ol>
  <li>Item 1</li>
  <li>Item 2</li>
</ol>
<!-- Tables -->
<table>
  <tr>
    <th>Header 1</th>
    <th>Header 2</th>
  </tr>
  <tr>
```

```
    <td>Cell 1</td>
    <td>Cell 2</td>
  </tr>
</table>
```

HTML Forms:

```
<!-- Form -->
<form action="submit-page.php" method="post">
  <!-- Form controls go here -->
</form>

<!-- Text Input -->
<input type="text" name="fieldName">

<!-- Checkbox -->
<input type="checkbox" name="fieldName" value="value">

<!-- Radio Buttons -->
<input type="radio" name="fieldName" value="value1">
<input type="radio" name="fieldName" value="value2">

<!-- Select Dropdown -->
<select name="fieldName">
```

```
  <option value="value1">Option 1</option>
  <option value="value2">Option 2</option>
</select>

<!-- Textarea -->
<textarea name="fieldName"></textarea>
<!-- Submit Button -->
<input type="submit" value="Submit">
```

HTML Semantic Elements:

```
<!-- Header -->
<header>
  <!-- Header content goes here -->
</header>

<!-- Navigation -->
<nav>
  <!-- Navigation links go here -->
</nav>
<!-- Main Content -->
<main>
  <!-- Main content goes here -->
</main>
```

```
<!-- Article -->
<article>
  <!-- Article content goes here -->
</article>

<!-- Section -->
<section>
  <!-- Section content goes here -->
</section>

<!-- Footer -->
<footer>
  <!-- Footer content goes here -->
</footer>
```

HTML Media Elements:

```
<!-- Video -->
<video src="video.mp4" controls></video>

<!-- Audio -->
<audio src="audio.mp3" controls></audio>
```

```
<!-- Embedding Content -->
<iframe src="https://example.com"></iframe>
<object data="file.pdf"></object>

<!-- Figure and Caption -->
<figure>
  <img src="image.jpg" alt="Image Description">
  <figcaption>Caption goes here</figcaption>
</figure>
```

HTML Attributes:

```
<!-- ID and Class -->
<div id="elementId" class="className"></div>
<!-- Style -->
<div style="property: value;"></div>
<!-- Width and Height -->
<img src="image.jpg" width="200" height="150">

<!-- Link Target -->
```

```
<a href="https://example.com" target="_blank">Link Text</a>

<!-- Accessibility -->
<img src="image.jpg" alt="Image Description" title="Image Title">

<!-- Data Attributes -->
<div data-custom="value"></div>
```

HTML Head Meta Tags:

```
<!-- Charset -->
<meta charset="UTF-8">

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

<!-- Page Title -->
<title>Page Title</title>

<!-- Description and Keywords -->
<meta name="description" content="Page description">
<meta name="keywords" content="keyword1, keyword2">
```

```
<!-- CSS Stylesheet -->
<link rel="stylesheet" href="styles.css">

<!-- JavaScript -->
<script src="script.js"></script>
```

HTML5 Features:

```
<!-- HTML5 Video -->
<video>
  <source src="video.mp4" type="video/mp4">
  Your browser does not support the video tag.
</video>


<!-- HTML5 Audio -->
<audio>
  <source src="audio.mp3" type="audio/mpeg">
  Your browser does not support the audio tag.
</audio>
```

```
<!-- HTML5 Canvas -->
<canvas></canvas>

<!-- HTML5 Geolocation -->
<button onclick="getLocation()">Get Location</button>

<!-- HTML5 Local Storage -->
<script>
  localStorage.setItem('key', 'value');
  const value = localStorage.getItem('key');
</script>
```

CSS Cheat Sheet

- `px` : Absolute Length
- `rem` : Relative to the `font-size` of the root Element
- `em` : Relative to the `font-size` of the Element
- `%` : Relative to the parent Element
- `vw` : Relative to the viewport's width, `1vw` = `1%` * viewport's width
- `vh` : Relative to the viewport's height, `1vh` = `1%` * viewport's height
- `vmin` : Relative to the viewport's smaller dimension, `1vmin` = min(`1vh` , `1vw`)
- `vmax` : Relative to the viewport's larger dimension, `1vmax` = max(`1vh` , `1vw`)
- `ch` : Relative to the width of the glyph "0" of the element's font
- `in` : Inches `1in` = `2.54cm` = `96px`
- `pc` : Picas `1pc` = `1in` / `6` = `16px`
- `pt` : Points `1pt` = `1in` / `72` = `1.333px` (approximately)
- `cm` : Centimeters `1cm` = `1in` / `2.54` = `37.8px` (approximately)
- `mm` : Millimeters `1mm` = `1cm` / `10` = `3.78px` (approximately)

CSS Flex Ultimate Cheatsheet

CONTAINER {PARENT} PROPERTIES

DISPLAY

Enables Flex For All Children

display: flex

display: inline-flex

JUSTIFY-CONTENT

Attempts to distribute extra space on main axis

flex-start

flex-end

center

space-between

space-evenly

space-around

ALIGN-ITEMS

Determine how items are laid out on the cross-axis.

flex-start

flex-end

center

baseline

stretch

ALIGN-CONTENT

Only has an effect with more than one line of content.

flex-start

flex-end

center

space-between

space-evenly

space-around

stretch

FLEX-DIRECTION

Establishes the main axis.

flex-direction: row

flex-direction: row-reverse

flex-direction: column

flex-direction: column-reverse

FLEX-WRAP

Wraps items if they can't be made to fit on one line.

flex-wrap: no-wrap

flex-wrap: wrap

flex-wrap: wrap-reverse

CONTAINER {PARENT} PROPERTIES

DISPLAY

Establishes a new grid formatting context for children.

display: grid

display: inline-grid

GRID-TEMPLATE

Defines the rows & columns of the grid.

grid-template-columns: 14px 14px 14px;
grid-template-rows: 14px 14px 14px;

grid-template-columns: repeat(3, 14px);
grid-template-rows: repeat(3, 14px)

grid-template-columns: 5px auto 5px;
grid-template-rows: 5px auto 5px;

grid-template-columns: 10% 10% auto;
grid-template-rows: 10% 10% auto;

GRID-GAP

Defines the size of column & row gutters

grid-gap: 14px;

grid-gap: 1px 14px;

grid-row-gap: 1px;
grid-column-gap: 14px ;

Note: You can also use gap which is very similar go grid [illegible] for both flexbox & grid.

JUSTIFY-CONTENT

Justifies all grid content on row axis when total grid size is smaller than container.

justify-content: start

justify-content: end

justify-content: center

justify-content: stretch

justify-content: space-around

justify-content: space-evenly

justify-content: space-between

ALIGN-CONTENT

Justifies all grid content on column axis when total grid size is smaller than container.

align-content: start

align-content: end

align-content: center

align-content: stretch

align-content: space-around

align-content: space-evenly

align-content: space-between

GRID-AUTO-FLOW

Algorithm for automatically placing grid items that aren't explictly placed.

tells the auto-placement algorithm to fill in each row in turn, adding new rows as necessary

grid-auto-flow: row

tells the auto-placement algorithm to fill in each column in turn, adding new columns as necessary

grid-auto-flow: column

tells the auto-placement algorithm to attempt to fill in holes earlier in the grid if smaller items come up later

grid-auto-flow: column

JUSTIFY-ITEMS

Aligns content in a grid item along the row axis.

justify-items: start

justify-items: end

justify-items: center

justify-items: stretch (default)

ALIGN-ITEMS

Aligns content in a grid item along the column axis.

align-items: start

align-items: end

align-items: center

align-items: stretch (default)

ITEM {CHILDREN} PROPERTIES

FLEX-GROW

Allows you to determine how each child is allowed to grow as a part of a whole.

flex-grow: 1;
(Applied to all items)

flex-grow: (1, 2 & 3)

FLEX-BASIS

Define the size of an element before remaining space is distributed.

first item 20%;
second item 40%

FLEX-SHRINK

Allows an item to shrink if necessary. Only really useful with a set size or flex-basis.

both want to be 100% wide, 2nd item has flex-shrink: 2

ALIGN-SELF

Sets alignment for individual item.

3rd item has
align-self: flex-end

ORDER

The order property specifies the order of the flex items.

order: -1; on 3rd item

ITEM {CHILDREN} PROPERTIES

GRID-ROW

Determines an items row-based location within the grid.

```
grid-row-start: 1;
grid-row-end: 3;
```

```
grid-row-start: span 3
```

```
grid-row-start: 2;
grid-row-end: 4;
```

```
grid-row: 1 / 3;
```

```
grid-row: 1 / span 3;
```

GRID-COLUMN

Determines an items column-based location within the grid.

```
grid-column-start: 1;
grid-column-end: 3;
```

```
grid-column-start: span 3;
```

```
grid-column-start: 2;
grid-column-end: 4;
```

```
grid-column: 2 / 3;
```

```
grid-row: 2 / span 2;
```

GRID-ROW + GRID-COLUMN

Combining grid rows with grid columns.

```
grid-row: 1 / spa
grid-column: 1 / span 2;
```

```
grid-row: 2 / span 2;
grid-column: 2 / span 2;
```

JUSTIFY-SELF

Aligns content for a specific grid item along the row axis.

```
justify-self: start;
```

```
justify-self: end;
```

```
justify-self: center;
```

```
justify-self: stretch;
```

ALIGN-SELF

Aligns content for a specific grid item along the column axis.

```
align-self: start;
```

```
align-self: end;
```

```
align-self: center;
```

```
align-self: stretch;
```

```
.selector {
   filter: blur("2px");
}
```

VALUE

PROPERTY

FUNCTION

Note: You can apply multiple functions but it has to be space seperated without comma.

No Filter Applied

filter: blur(2px);

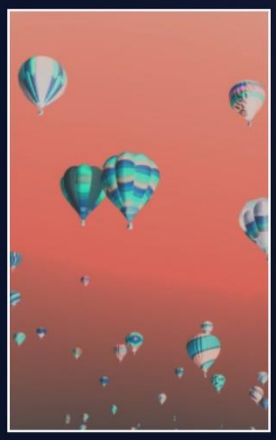
filter: brightness(0.4);

filter: contrast(200%);

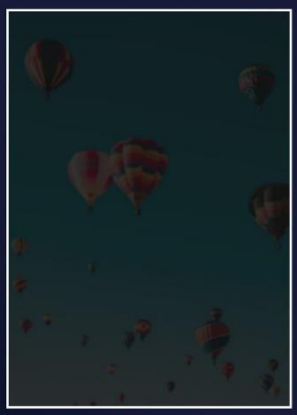
filter: drop-shadow(16px red);

filter: grayscale(80%);

filter: hue-rotate(90deg);



filter: invert(85%);



filter: opacity(15%);

filter: saturate(400%);

filter: sepia(560%);

ANIMATION SHORTHAND PROPERTY

```
animation:  dance  300ms  linear  100ms  infinite  alternate-reverse  both  reverse
            name   duration timing-function delay  count  direction  fill-mode play-state
```

ANIMATION NAME

- Defines which animation keyframe to use.

```
.box {
  background-color: purple;
  border-radius: 10px;
  width: 100px;
  height: 100px;

  /* Animation Name */
  animation-name: bounce;
}
```

- If no animation name is specified, no animation is played.
- If the name is specified, the keyframes matching the name will be used.

ANIMATION DURATION

- Defines how long the animation lasts.

```
.box {
  background-color: purple;
  border-radius: 10px;
  width: 100px;
  height: 100px;

  animation-name: bounce;
  /* Animation Duration */
  animation-duration: 2s;
}
```

- The default value is **zero seconds:** the animation will simply not play.
- You can use **decimal** values in **seconds** with keyword {s}
- You can use **milliseconds** instead of seconds, with the keyword {ms}

ANIMATION TIMING FUNCTION

- Defines how the values between start & the end of the animation are calculated.

```
.box {
  background-color: purple;
  border-radius: 10px;
  width: 100px;
  height: 100px;

  animation-name: bounce;
  animation-duration: 2s;
  /* Animation Timing Function */
  animation-timing-function: ease;
}
```

	Starts	Middle	Ends
ease:	Slow	acclearate	Slow
ease-in;	Slow	→	acclearate gradully
ease-out;	quickly	→	decelerates gradually
ease-in-out;	quickly	→	decelerates gradually
linear;	constant	constant	constant

ANIMATION DELAY

- Defines how long the animation has to wait before starting.

```
.box {
  background-color: purple;
  border-radius: 10px;
  width: 100px;
  height: 100px;

  animation-name: bounce;
  animation-duration: 2s;
  animation-timing-function: ease;
  /* Animation Delay */
  animation-delay: 2s;
}
```

animation-delay: **0s;**

The animation will wait **zero seconds,** and thus start right away.

animation-delay: **1.2s;**

You can use **decimal** values in **seconds** with the keyword {s}.

animation-delay: **2400ms;**

You can use **milliseconds** instead of seconds, with the keyword {ms}.

ANIMATION ITERATION COUNT

- Defines how many times the animation is played.

```css
.box {
  background-color: purple;
  border-radius: 10px;
  width: 100px;
  height: 100px;

  animation-name: bounce;
  animation-duration: 2s;
  animation-timing-function: ease;
  animation-delay: 2s;
  /* Animation Iteration Count */
  animation-iteration-count: 0;
}
```

`animation-iteration-count: 2s;`

You can use **integer values** to define a specific amount of times the animation will play.

`animation-iteration-count: infinite;`

By using the keyword **infinite,** the animation will play indefinitely.

ANIMATION DIRECTION

- Defines in which direction the animation is played.

```css
.box {
  background-color: purple;
  border-radius: 10px;
  width: 100px;
  height: 100px;

  animation-name: bounce;
  animation-duration: 2s;
  animation-timing-function: ease;
  animation-delay: 2s;
  animation-iteration-count: 0;
  /* Animation Direction */
  animation-direction: reverse;
}
```

`animation-direction: normal;`

Played **forwards**. When it reaches the end, it starts over at the first keyframe.

`animation-direction: reverse;`

Played **backwards:** begins at the last keyframe, finishes at the first keyframe.

`animation-direction: alternate;`

Played **forwards** first, then **backwards.**

`animation-direction: alternate-reverse;`

Played **backwards** first, then **forwards.**

ANIMATION FILL MODE

- Defines what happens before an animation starts and after it ends.

```css
.box {
  background-color: purple;
  border-radius: 10px;
  width: 100px;
  height: 100px;

  animation-name: bounce;
  animation-duration: 2s;
  animation-timing-function: ease;
  animation-delay: 2s;
  animation-iteration-count: 0;
  animation-direction: reverse;
  /* Animation Fill mode */
  animation-fill-mode: backwards;
}
```

`animation-fill-mode: none;`

The element is set to its default state before the animation **starts**, and returns to that default state after the animation **ends.**

`animation-fill-mode: forwards;`

The last styles applied at the end of the animation are retained **afterwards.**

`animation-fill-mode: backwards;`

The animation's styles will already be applied before the animation actually **starts.**

`animation-fill-mode: both;`

The styles are applied before and after the **animation plays.**

ANIMATION PLAY STATE

- Defines if an animation is playing or not.

```css
.box {
  background-color: purple;
  border-radius: 10px;
  width: 100px;
  height: 100px;

  animation-name: bounce;
  animation-duration: 2s;
  animation-timing-function: ease;
  animation-delay: 2s;
  animation-iteration-count: 0;
  animation-direction: reverse;
  animation-fill-mode: backwards;
  /* Animation Play State */
  animation-play-state: paused;
}
```

`animation-play-state: running;`

If the **animation-duration** and **animation-name** are defined, the animation will start playing automatically.

`animation-play-state: paused;`

The animation is set **paused** at the first keyframe.

TRANSFORM FUNCTIONS

- matrix()
- matrix3d()
- perspective()
- rotate()
- rotate3d()
- rotatex()
- rotatey()
- rotatez()
- scale()
- scalex()
- scaley()
- scalez()
- skew()
- skewx()
- skewy()
- translate()
- translate3d()
- translatex()
- translatey()
- translatez()

REFERENCE FUNCTIONS

- attr()
- env()
- url()
- var()

SHAPE FUNCTIONS

- circle()
- ellipse()
- inset()
- polygon()
- path()

MATH FUNCTIONS

- calc()
- clamp()
- max()
- min()
- abs()
- acos()
- asin()
- atan()
- atan2()
- cos()
- exp()
- hypot()
- log()
- mod()
- pow()
- rem()
- round()
- sign()
- sin()
- sqrt()
- tan()

IMAGE FUNCTIONS

- conic-gradient()
- linear-gradient()
- radial-gradient()
- repeating-linear-gradient()
- repeating-radial-gradient()
- repeating-conic-gradient()
- cross-fade()
- element()
- paint()

FILTER FUNCTIONS

- blur()
- brightness()
- contrast()
- drop-shadow()
- grayscale()
- hue-rotate()
- invert()
- opacity()
- saturate()
- sepia()

FONT FUNCTIONS

- stylistic()
- styleset()
- character-varient()
- swash()
- ornaments()
- annotation()

COLOR FUNCTIONS

- hsl()
- hsla()
- hwb()
- lab()
- lch()
- rgb()
- rgba()
- color()
- color-mix()
- color-contrast()
- device-cmyk()

COUNTER FUNCTIONS

- counter()
- counters()
- symbols()

GRID FUNCTIONS

- fit-content()
- minmax()
- repeat()

ALL CSS SELECTOR TYPES EXPLAINED

* : Universal Selector

- Selects all elements of any type on HTML document.

```css
* {
  color: blue;
}
```

This rule will change every HTML element on the page to have blue text.

type : type Selector

- Selects elements by node name, selects all elements of the given type within a given document. **Ex: div, span**

```css
/* All <a> elements. */
a {
  color: red;
}
```

This rule will change every anchor element on the page to have red color text.

.class-name : Class Selector

- Selects elements based on the content of **class=""** attribute

```css
/* All elements with class="spacious" */
.spacious {
  margin: 2em;
}
```

This rule will change add margin of 2em to every element with class attribute of spacious

#id-name : Id Selector

- Selects an element based on the value of its **id=""** attribute.

```css
/* The element with id="demo" */
#demo {
  padding: 2em;
}
```

This rule will change add padding of 2em to the element with id attribute of demo

[attr] : Attribute Selector

- matches elements based on the presence or value of a **given attribute.**

```css
/* Select <a> element with a title attribute */
a [title] {
  color: blue;
}
```

This rule will change the color of <a> element with title attribute to blue.

Grouping Selectors

- Grouping selector **(,)** can select multiple selector at once.

```css
/* Selects the div & span and change
                    their color to red */
div,
span {
  color: red;
}
```

This rule will change the color of & element to red.

Compound Selectors

- We can combine selectors to increase specificity & readability.

```css
/* Only select the <a> element which also
                    has a class of spacious. */
a.spacious {
  color: red;
}
```

This rule will change the color of only a element which also has a class of spacious.

ALL CSS COMBINATORS

Descendant Combinator

- A Descendant Combinator allows us to target a child element.

Select all child & grandchilds of a parent.

```css
/* Selects all <strong> elements that
                    are child of p */
p strong {
  color: red;
}
```

This rule will change the color of elements that are child of <p> to blue.

Child Combinator

- Child Combinator helps us to **select direct child of the parent.**

Selects only direct children & not the grandchildren of a parent.

```css
/* Selects all <strong> elements that
                are direct child of p */
p > strong {
  color: blue;
}
```

This rule will change the color of elements that are direct child of <p> to blue.

General sibling Combinator

- A general sibling works in a way that even if the **sibling before the child doesn't the same it will still select it.**

```css
/* Selects <strong> elements even if there is
someting different before last <strong> of <p>
it will still select it */
p ~ strong {
  color: red;
}
```

This rule will change all the elements even if the element before the child is not same.

Adjacent Sibling Combinator

- Adjacent sibling works in a way that it will **only select a child matches the first element before it.**

```css
/* Selects <strong> elements only if it
        matches the first element before it */
p + strong {
  color: blue;
}
```

This rule will change the color of elements only if matches the first element before it of <p>

PSEUDO CLASSES & ELEMENTS SELECTOR:-

: Pseudo Classes

- The **:** pseudo allow the selection of elements based on **state information** that not contained in the **document tree.**

Ex: **a:visited**

```css
/* Any button over which the user's
      pointer is hovering will change
            its background to blue */
button:hover {
  color: blue;
}
```

This rule will change the color of <button> element to blue when user pointer is hovered

:: Pseudo Elements

- The **::** pseudo represent entities that are not included in HTML

Ex: **p::first-line**

```css
/* The first line of every <p> element. */
p::first-line {
  color: blue;
}
```

This rule will change the color of every first line of p element to blue

JavaScript cheat sheet

Variables and Data Types:

```
// Variable declaration and assignment
let variableName = value;

// Data Types
let num = 10;           // Number
let str = "Hello";      // String
let bool = true;        // Boolean
let arr = [1, 2, 3];    // Array
let obj = {key: 'value'}; // Object
let func = function() {}; // Function
```

String Manipulation:

```
// Concatenation
const str1 = 'Hello';
const str2 = 'World';
const result = str1 + ' ' + str2; // "Hello World"
```

```
// String methods
const str = 'Hello, World!';
str.length;               // Length of the string
str.toLowerCase();        // Convert to lowercase
str.toUpperCase();        // Convert to uppercase
str.indexOf('o');         // Index of first occurrence of 'o'
str.lastIndexOf('o');     // Index of last occurrence of 'o'
str.includes('World');    // Check if 'World' is present
str.slice(start, end);    // Extract a portion of the string
str.substring(start, end); // Extract a portion of the string
str.replace('Hello', 'Hi');// Replace 'Hello' with 'Hi'
str.split(',');           // Split the string into an array
str.trim();               // Remove leading/trailing whitespace
```

Operators:

```
// Arithmetic operators
let sum = 2 + 3;
let difference = 5 - 2;
let product = 4 * 6;
let quotient = 10 / 2;
let remainder = 10 % 3;
// Comparison operators
let isEqual = 5 === '5'; // false (strict equality)
let isNotEqual = 5 !== 3; // true
let greaterThan = 10 > 5; // true
let lessThan = 2 < 4;     // true
```

```
let greaterOrEqual = 5 >= 5; // true
let lessOrEqual = 3 <= 3; // true

// Logical operators
let andOp = true && false; // false (logical AND)
let orOp = true || false;  // true (logical OR)
let notOp = !true;         // false (logical NOT)
// Assignment operators
let x = 5;
x += 3; // x = x + 3
x -= 2; // x = x - 2
x *= 4; // x = x * 4
x /= 2; // x = x / 2
```

Conditionals:

```
// If-else statement
if (condition) {
  // Code block executes when the condition is true
} else {
  // Code block executes when the condition is false
}
// Ternary operator (shorter if-else)
let result = (condition) ? 'true case' : 'false case';
// Switch statement
switch (value) {
  case 1:
    // Code block for case 1
```

```
    break;
  case 2:
    // Code block for case 2
    break;
  default:
    // Code block if no cases match
    break;
}
```

Loops:

```
// For loop
for (let i = 0; i < 5; i++) {
  // Code block executes 5 times with i values 0 to 4
}
// While loop
let count = 0;
while (count < 5) {
  // Code block executes as long as the condition is true
  count++;
}
// Do-While loop
let num = 0;
do {
  // Code block executes at least once, then checks the condition
  num++;
} while (num < 5);
// For...of loop (for arrays and iterable objects)
```

```
const array = [1, 2, 3];
for (const element of array) {
 console.log(element);  }
```

Functions:

```
// Function declaration
function functionName(parameters) {
  // Code block
  return value; // Optional return statement
}
// Function expression (anonymous function)
const functionName = function(parameters) {
  // Code block
};
// Arrow function (ES6+)
const functionName = (parameters) => {
  // Code block
};
```

Arrays:

```
// Array declaration and initialization
let arrayName = [element1, element2, element3];
// Accessing array elements
let element = arrayName[index]; // Index starts from 0
// Modifying array elements
arrayName[index] = newValue;
// Array methods
arrayName.push(element);        // Add element at the end
```

```
arrayName.pop();                // Remove element from the end
arrayName.shift();              // Remove element from the beginning
arrayName.unshift(element);     // Add element at the beginning
arrayName.slice(start, end);    // Extracts a portion of the array
arrayName.splice(index, count); // Remove or replace elements
arrayName.length;               // Length of the array
```

Objects:

```
// Object declaration and initialization
let objectName = {
  key1: value1,
  key2: value2,
};

// Accessing object properties
let value = objectName.key;

// Modifying object properties
objectName.key = newValue;

// Object methods
Object.keys(objectName);    // Returns an array of keys
Object.values(objectName);  // Returns an array of values
Object.entries(objectName); // Returns an array of key-value pairs
```

DOM Manipulation:

```
// Accessing elements
const element = document.getElementById('elementId');
const elements = document.getElementsByClassName('className');
const elements = document.getElementsByTagName('tagName');
const element = document.querySelector('selector');
const elements = document.querySelectorAll('selector');
// Modifying element content
element.textContent = 'New text';
element.innerHTML = '<b>Bold text</b>';

// Modifying element attributes
element.setAttribute('attribute', 'value');
element.getAttribute('attribute');
element.removeAttribute('attribute');

// Adding and removing classes
element.classList.add('className');
element.classList.remove('className');
element.classList.toggle('className');
element.classList.contains('className');

// Event handling
element.addEventListener('eventName', eventHandler);
element.removeEventListener('eventName', eventHandler);
```

AJAX and Fetch:

```
// XMLHttpRequest (XHR)
const xhr = new XMLHttpRequest();
xhr.open('GET', 'url', true);
xhr.onreadystatechange = function() {
  if (xhr.readyState === 4 && xhr.status === 200) {
    const response = JSON.parse(xhr.responseText);
    // Process the response
  }
};
xhr.send();

// Fetch API
fetch('url')
  .then(response => response.json())
  .then(data => {
    // Process the data
  })
  .catch(error => {
    // Handle the error
  });
```

DOM Events:

```
// Common DOM events
element.addEventListener('click', eventHandler);
element.addEventListener('mouseover', eventHandler);
element.addEventListener('keydown', eventHandler);
element.addEventListener('submit', eventHandler);
// Event propagation and delegation
event.stopPropagation();
event.preventDefault();
// Event delegation
document.addEventListener('eventName', function(event) {
  const target = event.target;
  if (target.matches('selector')) {
    // Handle the event
  }
});
```

Cookies and Local Storage:

```
// Cookies
```

```
document.cookie = 'name=value; expires=date; path=path; domain=domain; secure';
// Local Storage
localStorage.setItem('key', 'value');
localStorage.getItem('key');
localStorage.removeItem('key');
localStorage.clear();
```

Error Handling:

```
// Try-catch block
try {
  // Code that might throw an error
} catch (error) {
  // Handle the error
}

// Error types
new Error('Error message');
new TypeError('Type error');
new RangeError('Range error');
```

Promises and Asynchronous Programming:

```
// Promises
const promise = new Promise((resolve, reject) => {
  // Asynchronous operation
  if (success) {
    resolve(data);
  } else {
    reject(error);
  }
});
promise.then(data => {
  // Handle the resolved data
}).catch(error => {
  // Handle the rejected error
});
// Async/await
async function asyncFunction() {
  try {
    const data = await promise;
    // Handle the resolved data
  } catch (error) {
    // Handle the rejected error
  }
}
```

Regular Expressions:

```
// Regular expression creation
```

```
const regex = /pattern/;
const regex = new RegExp('pattern');

// Matching and testing
regex.test('string');
'string'.match(regex);

// Replacement
const newString = 'string'.replace(regex, 'replacement');

// Flags
const regex = /pattern/gi;
// global, case-insensitive
```

Manipulating Styles:

```
// Modifying inline styles
element.style.property = 'value';
// Modifying classes
element.classList.add('className');
element.classList.remove('className');
element.classList.toggle('className');
element.classList.contains('className');

// Getting computed styles
const styles = getComputedStyle(element);
const value = styles.getPropertyValue('property');
```

Timers:

```
// setTimeout
const timeoutId = setTimeout(function() {
  // Code to execute after a delay
}, delayInMilliseconds);
clearTimeout(timeoutId);

// setInterval
const intervalId = setInterval(function() {
  // Code to execute repeatedly with a fixed interval
}, intervalInMilliseconds);

clearInterval(intervalId);
```

Date and Time:

```
// Creating Date objects
const currentDate = new Date();
const specificDate = new Date('2022-01-01');
const timestamp = new Date(1630444800000);

// Date methods
currentDate.getFullYear();   // Get the year (e.g., 2023)
currentDate.getMonth();      // Get the month (0-11)
currentDate.getDate();       // Get the day of the month (1-31)
currentDate.getHours();      // Get the hour (0-23)
currentDate.getMinutes();    // Get the minutes (0-59)
currentDate.getSeconds();    // Get the seconds (0-59)
currentDate.getTime();       // Get the timestamp (milliseconds)
currentDate.toLocaleString(); // Convert to local string format
```

HAPPY CODING

CODERSTAR

FOLLOW