KOSHYS INSTITUTE OF MANAGEMENT STUDIES

Computer multimedia and animation

Bachelor of computer application

4^{th} semester

Mr.IRFAN.J

Course Code:	21BCA3C11L
Course Title:	Computer Multimedia and Animation
Formative Assessment marks:	40 marks
Summative Assessment Marks:	60 marks

UNIT 1:

Web Design: Origins and evolution of HTML, Basic syntax, Basic text markup, Images, Lists, Tables, Forms, Frame, Overview and features of HTML5. CSS: Introduction, Levels of style sheets, Style specification formats, Selector forms, Property value forms, Font properties, List properties, Color, Alignment of text, The and tags; Overview and features of CSS3. JavaScript: Object orientation and JavaScript; General syntactic characteristics; Primitives, operations, and expressions; Screen output and keyboard input

UNIT 2:

Animation: What is an Animation? The Start and End States, Interpolation, Animations in HTML. All About CSS Animations, Creating a Simple Animation, Detailed Look at the CSS Animation Property, Key frames, Declaring Multiple Animations, Wrap-up. All about CSS Transitions, Adding a Transition, Looking at Transitions in Detail, the Longhand Properties, Longhand Properties vs. Shorthand Properties, Working with Multiple Transitions

UNIT 3:

HTML5 – SVG: Viewing SVG Files, Embedding SVG in HTML5, HTML5 − SVG Circle, HTML5 − SVG Rectangle, HTML5 − SVG Line, HTML5 − SVG Ellipse, HTML5 − SVG Polygon, HTML5 − SVG Polyline, HTML5 − SVG Gradients, HTML5 − SVG Star.

UNIT 4:

HTML5 – CANVAS: The Rendering Context, Browser Support, HTML5 Canvas Examples, Canvas - Drawing Rectangles, Canvas - Drawing Paths, Canvas - Drawing Lines, Canvas - Drawing Bezier Curves, Canvas - Drawing Quadratic Curves, Canvas - Using Images, Canvas - Create Gradients.

UNIT 5:

HTML5 - Styles and Colors, Canvas - Text and Fonts, Canvas - Pattern and Shadow, Canvas - Save and Restore States, Canvas - Translation, Canvas - Rotation, Canvas - Scaling, Canvas - Transforms, HTML5 Canvas - Composition, Canvas – Animations..

UNIT 1
WEB DESIGN

Web Design: Origins and evolution of HTML, Basic syntax, Basic text markup, Images, Lists, Tables, Forms, Frame, Overview and features of HTML5. CSS: Introduction, Levels of style sheets, Style specification formats, Selector forms, Property value forms, Font properties, List properties, Color, Alignment of text, The and tags; Overview and features of CSS3. JavaScript: Object orientation and JavaScript; General syntactic characteristics; Primitives, operations, and expressions; Screen output and keyboard input

WEB DESIGN

- Web design encompasses the process of creating and designing the visual appearance and user experience of websites.
- It involves various elements such as layout, colors, typography, graphics, and interactive features to enhance the aesthetics and functionality of a website.

ORIGINS AND EVOLUTION OF HTML

HTML STRUCTURE

- HTML (Hypertext Markup Language) provides a structured way to define the elements and content of a web page.
- The structure of an HTML document consists of several key components:

SL .No	Element	Definition
1	**Document Type Declaration (DTD)**	➢ The DTD defines the version of HTML being used in the document. It is typically declared at the beginning of an HTML file using the <!DOCTYPE> tag.
2	**<html>**	➢ The <html> element is the root element of an HTML document. It encapsulates the entire document and contains two main sections: the <head> and the <body>.
3	**<head>**	➢ The <head> element provides metadata and other non-visible information about the document. ➢ It includes elements such as the document title (<title>), character encoding (<meta charset>), CSS stylesheets (<link rel="stylesheet">), JavaScript files (<script>), and more.
4	**<body>**	➢ The <body> element contains the visible content of the web page. ➢ It includes elements such as headings (<h1>, <h2>, etc.), paragraphs (<p>), images (<img>), links (<a>), lists (, ,), tables (<table>, <tr>, <td>), forms (<form>, <input>, <button>), and other structural elements.
5	**Comments**	➢ Comments can be added to an HTML document to provide explanatory or informational notes to developers. ➢ They are written within <!-- and --> tags and are not displayed on the web page.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>My Web Page</title>
   <link rel="stylesheet" href="styles.css">
   <script src="script.js"></script>
</head>
<body>
   <h1>Welcome to My Web Page</h1>
   <p>This is a paragraph of text.</p>
   <img src="image.jpg" alt="Image description">
   <a href="https://www.example.com">Link</a>
</body>
</html>
```

BASIC SYNTAX

- When it comes to web development and programming, different languages have their own syntax rules.
- Here are the basic syntax rules for a few popular languages:

SL.No	Language	Definition	Example
1	**HTML (Hypertext Markup Language)**	➢ HTML uses tags to define the structure and content of a web page. ➢ Tags are written within angle brackets (< >) and are typically used in pairs (opening and closing tags).	<tagname>Content goes here</tagname>
2	**CSS (Cascading Style Sheets)**	➢ CSS is used to style and format the appearance of HTML elements. ➢ CSS rules consist of a selector (identifies the element to style) and a declaration block (defines the styles).	selector { property: value; }
3	**JavaScript**	➢ JavaScript is a programming language used for client-side and server-side scripting. ➢ JavaScript statements are typically terminated by a semicolon (;).	var variableName = value;
4	**Python**	➢ Python is a popular high-level programming language known for its readability and simplicity. ➢ Indentation is crucial in Python, as it determines the structure of code blocks (e.g., loops, conditionals).	variable_name = value if condition: # Code block statement

5	**Java**	➢ Java is an object-oriented programming language widely used for building applications.
		➢ Java code is organized into classes and methods.
		➢ Statements are terminated with a semicolon (;).
			public class MyClass {
			public static void main(String[] args) {
			// Code block
			int variableName = value;
			}
			}

6	**PHP**	➢ PHP is a server-side scripting language commonly used for web development.
		➢ PHP code is embedded within HTML, typically enclosed in <?php ?> tags.
			<?php
			$variableName = value;
			if ($condition) {
			// Code block
			echo "Statement";
			}
			?>

BASIC TEXT MARKUP

- Basic text markup refers to the use of formatting and styling techniques to enhance the appearance and structure of text.
- Markup languages like HTML provide tags or syntax to apply formatting to text.
- Here are some commonly used basic text markup techniques:

SL.No	Element	Definition	Example
1	**Headings**	➢ HTML: Use <h1> to <h6> tags for different heading levels.	This is a Heading 1
2	**Paragraphs**	➢ HTML: Use tag to define paragraphs.	This is a paragraph of text.
3	**Bold**	➢ HTML: Use or <b> tags to make text bold.	This text is bold.
4	**Italic**	➢ HTML: Use or <i> tags to italicize text.	This text is italicized.
5	**Underline**	➢ HTML: Use <u> tag to underline text.	<u>This text is underlined.</u>
6	**Strikethrough**	➢ HTML: Use <s> or <del> tags to create strikethrough text.	<s>This text is strikethrough.</s>
7	**Subscript and Superscript**	➢ HTML: Use <sub> and <sup> tags for subscript and superscript text, respectively.	This is a _{subscript} and this is a ^{superscript}.
8	**Links**	➢ HTML: Use <a> tag to create hyperlinks.	Link
9	**Blockquotes**	➢ HTML: Use <blockquote> tag to create blockquotes for quoted text.	<blockquote>This is a blockquote.</blockquote>

IMAGES, LISTS, TABLES, FORMS TAGS

SL.No	Element	Definition	Example
1	**Images**	➢ HTML: Use the <img> tag to insert images into a web page. ➢ The src attribute specifies the image file URL, and the alt attribute provides alternative text for accessibility.	<img src="image.jpg" alt="Image description">
2	**Lists**	➢ HTML: Use for unordered (bullet) lists and for ordered (numbered) lists. List items are defined with tags.	 - Item 1 - Item 2

3	**Tables**	➢ HTML: Use the <table> tag to create tables, and use <tr> for table rows, <th> for table headers, and <td> for table data cells.	<table> <tr> <th>Header 1</th> </tr> <tr> <td>Data 1</td> </tr> </table>
4	**Forms**	➢ HTML: Use the <form> tag to create a form, and various input elements like <input>, <textarea>, and <select> for form fields. ➢ The action attribute specifies the URL to which the form data will be submitted, and the method attribute defines the HTTP method (e.g., GET or POST).	<form action="/submit" method="post"> <label for="name">Name:</label> <input type="text" id="name" name="name"> <input type="submit" value="Submit"> </form>

FRAMES TAG

- HTML: Frames are not commonly used in modern web design due to accessibility and usability issues.
- Instead, techniques like CSS and JavaScript are used for layout and content organization.
- However, if needed, frames can be created using the **<frame> or <iframe>** tags.
- The src attribute specifies the source URL of the content to be displayed within the frame.

OVERVIEW AND FEATURES OF HTML5

- HTML5 is the latest version of the Hypertext Markup Language, which is the standard markup language for creating and structuring content on the web.
- HTML5 introduces several new features and enhancements that improve the functionality, multimedia support, and interactivity of web pages.

Key features and capabilities of HTML5:

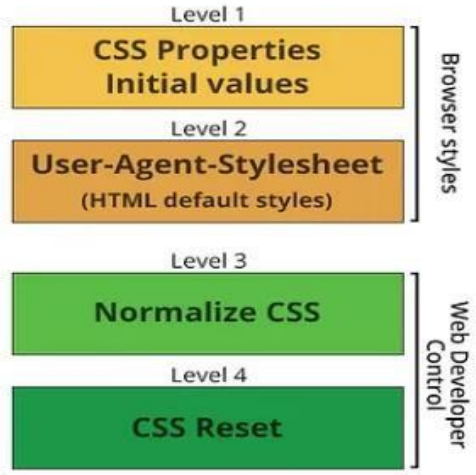
SL.No	Features	Definition
1	**Semantic Elements**	➢ HTML5 introduces a set of semantic elements that provide more meaningful structure and help describe the content of a webpage. ➢ These elements include **<header>**, **<nav>**, **<article>**, **<section>**, **<aside>**, **<footer>**, and more.
2	**Multimedia Support**	➢ HTML5 includes native support for embedding multimedia content without the need for third-party plugins like Adobe Flash. ➢ The **<video>** and **<audio>** elements allow developers to embed video and audio content directly into web pages, with support for different formats and customization options.

3	**Canvas**	➢ The <**canvas**> element in HTML5 provides a powerful and versatile drawing surface for rendering graphics, animations, and interactive visualizations directly within a web page. ➢ It enables the creation of dynamic and interactive content using JavaScript.
4	**Geolocation**	➢ HTML5 provides a Geolocation API that allows websites to access and utilize location information of a user's device, if permitted by the user. ➢ This feature enables applications such as location-based services, mapping, and real-time tracking.
5	**Forms Enhancements**	➢ New input types such as **<input type="date">**, **<input type="email">**, and **<input type="number">** provide specialized input fields with built-in validation. ➢ The **<datalist>** element allows the specification of a list of predefined options for input fields.
6	**Offline Web Applications**	➢ HTML5 includes the ability to create offline web applications that can function even when the user is not connected to the internet. ➢ The Application Cache API allows developers to specify which resources should be cached locally, enabling offline access and faster load times.
7	**Web Storage**	➢ HTML5 provides two mechanisms for client-side storage: localStorage and sessionStorage. ➢ These APIs allow web applications to store data locally on the user's device, providing a way to persist data across sessions or temporarily store data during a browsing session.
8	**Drag and Drop**	➢ HTML5 introduces native support for drag and drop interactions. ➢ This feature enables users to drag elements on a web page and drop them into designated areas, allowing for more intuitive userinteractions and data transfer.
9	**Responsive Images**	➢ HTML5 includes the **<picture>** and **<source>** elements, which allow developers to specify multiple image sources and automatically serve the most appropriate image based on factors such as screen size, resolution, and device capabilities. ➢ This helps optimize the loading and display of images across different devices and network conditions.
10	**Web Workers**	➢ HTML5 introduces Web Workers, which are background scripts that run separately from the main web page, allowing for parallel processing and better performance. ➢ Web Workers enable the execution of complex tasks without blocking the main user interface, improving the responsiveness and interactivity of web applications.

CSS

- CSS (Cascading Style Sheets) is a stylesheet language used to describe the presentation and visual appearance of HTML and XML documents.
- It allows web developers to control the layout, colors, fonts, and other visual aspects of web pages, separate from the underlying structure and content.
- CSS works by selecting HTML elements and applying styles to them using various properties and values.
- Styles can be applied directly to HTML elements using inline styles, included within the HTML file using internal stylesheets, or linked externally using external stylesheets.

Levels of CSS



SL.No	Level	Definition
1	**CSS1**	➢ The first level of CSS, introduced in 1996, provided basic styling capabilities such as font and color properties, text alignment, and margins. ➢ CSS1 laid the foundation for subsequent CSS versions.
2	**CSS2**	➢ Released in 1998, CSS2 expanded on CSS1 with additional features, including positioning and layout properties, support for media types (print, screen, etc.), and improved selector options. ➢ CSS2 also introduced the concept of pseudo-classes and pseudo-elements.
3	**CSS3**	➢ CSS3 is not a single version but rather a collection of separate modules that are being developed and released independently. ➢ CSS3 introduces numerous new features, enhancements, and capabilities that further extend the styling capabilities of CSS. Some notable features of CSS3 include border-radius, box-shadow, gradients, transitions, animations, media queries, and more.

4	**CSS4**	➢ While there is no official CSS4 specification, the term "CSS4" is sometimes used to refer to ongoing developments and proposals for future versions of CSS beyond CSS3. ➢ New features and modules are being proposed and implemented as separate modules rather than a single monolithic version.

STYLE SPECIFICATION FORMATS

- There are multiple formats and methodologies available for specifying styles in CSS.
- Here are some commonly used style specification formats:

SL.No	Selector	Definition	Example
1	**Inline Styles**	➢ Inline styles are applied directly to HTML elements using the style attribute. CSS properties and values are specified within the attribute's value. ➢ Inline styles have high specificity but can result in code duplication if used extensively.	This is a paragraph.

2	**Internal Stylesheets**	➢ Internal stylesheets are defined within the <style> tags within the <head> section of an HTML document. CSS rules are written directly within the <style> block
		➢ Internal stylesheets allow you to define styles for specific HTML elements within the same HTML file.
			<head>
			<style>
			p {
			color: blue;
			font-size: 16px;
			}
			</style>
			</head>
			<body>
			 This is a paragraph.
			</body>

3	**External Stylesheets**	➢ External stylesheets are separate CSS files linked to HTML documents using the <link> tag. The CSS rules are written in the external CSS file, which is referenced in the href attribute of the <link> tag ➢ External stylesheets promote separation of concerns by keeping styles separate from HTML, allowing for easier maintenance and reuse of styles across multiple HTML files.	<head> <link rel="stylesheet" href="styles.css"> </head> <body> This is a paragraph. </body>

4	**CSS Preprocessors**	➢ CSS preprocessors, such as Sass (Syntactically Awesome Style Sheets) and Less, introduce additional features and functionality to CSS. ➢ They use their own syntax and provide features like variables, nesting, mixins, functions, and more. ➢ Preprocessors transform the preprocessed code into standard CSS, which is then used in the HTML documents.
5	**CSS Frameworks**	➢ CSS frameworks like Bootstrap, Foundation, and Bulma offer pre-designed styles, layout systems, and components.

SELECTOR FORMS

SL.No	Selector	Definition	Example
1	**Element Selector**	➢ Selects elements based on their HTML tag name.	p selects all <p> elements.
2	**Class Selector**	➢ Selects elements based on the value of their class attribute	.my-class selects all elements with class="my-class"
3	**ID Selector**	➢ Selects an element based on the value of its id attribute.	#my-id selects the element with id="my-id".
4	**Attribute Selector**	➢ Selects elements based on the presence or value of an attribute.	[type="submit"] selects all elements with type="submit".

PROPERTY VALUE FORMS

SL.No	Values	Definition	Example
1	**Length Units**	➢ Specifies a measurement value.	px for pixels, em for relative to the font size of the element, rem for relative to the root font size.
2	**Color Values**	➢ Specifies colors.	#RRGGBB or #RGB for hexadecimal colors, rgb(r, g, b) for RGB values, rgba(r, g, b, a) for RGB values with alpha (transparency) channel.
3	**Percentage Values**	➢ Specifies a percentage relative to a parent value.	width: 50% sets the width to 50% of the parent container.
4	**URL Values**	➢ Specifies a URL.	background-image: url('image.jpg') sets the background image using the specified URL.

FONT PROPERTIES

SL.No	Attribute	Definition
1	**font-family**	➢ Specifies the font family or a list of fonts for text.
2	**font-size**	➢ Specifies the size of the font.
3	**font-weight**	➢ Specifies the thickness of the font.
4	**font-style**	➢ Specifies the style of the font (normal, italic, oblique).
5	**text-decoration**	➢ Specifies decorations like underline, overline, line-through.
6	**text-align**	➢ Specifies the horizontal alignment of text (left, right, center, justify).

LIST PROPERTIES

SL.No	Attribute	Definition
1	**list-style-type**	➢ Specifies the type of list marker (disc, decimal, square, etc.).
2	**list-style-image**	➢ Specifies an image as the list marker.
3	**list-style-position**	➢ Specifies the position of the list marker (inside or outside the list item).

COLOR

SL.No	Attribute	Definition
1	**color**	➢ Specifies the text color.
2	**background-color**	➢ Specifies the background color.
3	**border-color**	➢ Specifies the color of borders.
4	**opacity**	➢ Specifies the opacity of an element (0.0 to 1.0).

ALIGNMENT OF TEXT

SL.No	Attribute	Definition
1	**text-align**	➢ Specifies the horizontal alignment of text (left, right, center, justify).
2	**vertical-align**	➢ Specifies the vertical alignment of inline elements (baseline, top, middle, bottom).
3	**line-height**	➢ Specifies the height of a line of text, affecting vertical spacing.

AND TAGS

- The and tags are two commonly used HTML tags that serve different purposes in structuring and styling web content
- Difference between the and tags lies in their default behavior and intended usage.
- is an inline element used for small, inline-level styling or grouping,
- While is a block-level element used for larger sections, layout structures, or grouping of content.

SL.No	Tag	Definition	Example
1		➢ The tag is an inline element used to apply styles or add hooks to specific portions of text or inline elements within a larger block of content. ➢ It does not create a line break and is typically used for small, inline-level styling or grouping of elements. ➢ It does not inherently have any visual or semantic meaning and is often used in conjunction with CSS to apply styles or JavaScript to add functionality.	This text is highlighted.
2		➢ The tag is a block-level element used to group and organize larger sections of content or to create layout structures within a web page. ➢ It is a versatile container that does not have any inherent semantic meaning, allowing developers to apply custom styling or manipulate the content within it. ➢ It can contain other block-level or inline elements, and multiple tags can be nested to create complex layouts.	 Title Paragraph of text. Another paragraph.

JAVA SCRIPT

- JavaScript is a high-level programming language primarily used for client-side web development.
- It allows developers to add interactivity, dynamic behavior, and functionality to web pages.

OBJECT ORIENTATION AND JAVASCRIPT

- Java Script is an object-oriented programming language, which means it supports the concepts of objects, classes, and inheritance.

Key points regarding object orientation in JavaScript:

1. Objects and Properties
2. Classes and Prototypes
3. ECMAScript 2015 (ES6) Classes
4. Inheritance
5. Encapsulation and Abstraction
6. Polymorphism and Dynamic Typing

GENERAL SYNTACTIC CHARACTERISTICS

- JavaScript has several syntactic characteristics that define its structure and coding conventions.
- Here are some general syntactic characteristics of JavaScript:

SL.No	Attribute	Definition
1	**Case Sensitivity**	➢ JavaScript is case-sensitive, meaning that myVariable and myvariable are considered as different variables.
2	**Statements and Semicolons**	➢ JavaScript code consists of statements, which are individual instructions or commands. ➢ Statements in JavaScript are typically terminated with a semicolon (;), although it is not always required. ➢ JavaScript has automatic semicolon insertion (ASI) rules that insert semicolons in certain cases,
3	**Comments**	➢ JavaScript supports single-line comments, which are denoted by //, and multi-line comments, which are enclosed between /* and */. ➢ Comments are used to add explanatory
4	**Variables**	➢ Variables in JavaScript are declared using the var, let, or const keywords. ➢ The var keyword is used for declaring variables with function scope.
5	**Data Types**	➢ JavaScript has several built-in data types, including Primitive types & Complex types
6	**Operators**	➢ JavaScript supports various operators for performing arithmetic, comparison, logical, assignment, and other operations. ➢ Examples of operators include +, -, *, / for arithmetic, ==, ===, !=, !== for comparison, &&, ||, ! for logical operations, and =, +=, -= for assignment.
7	**Control Flow**	➢ JavaScript provides control flow structures such as if statements, for and while loops, switch statements, and conditional (ternary) operators (condition ? expression1 : expression2).
8	**Functions**	➢ JavaScript functions are declared using the function keyword, followed by a name (optional for anonymous functions), parentheses for parameters, and curly braces for the function body. ➢ Functions can be assigned to variables, passed as arguments to other functions, and returned as values from other functions.

PRIMITIVES

- In JavaScript, there are several primitive data types that are used to represent simple values. These primitive types are:

SL.No	Type	Definition
1	**Number**	➢ The number type represents numeric values.
2	**String**	➢ The string type represents textual data. ➢ Strings are enclosed in single quotes (') or double quotes ("),
3	**Boolean**	➢ The boolean type represents logical values. ➢ It has two possible values: true and false
4	**Null**	➢ The null type represents the absence of any object value.
	Undefined	➢ The undefined type represents a variable that has been declared but has not been assigned a value.

Example
```
let numberValue = 42;//  number
let stringValue = "Hello"; // string
let booleanValue = true; // boolean
let nullValue = null;     // null
let undefinedValue = undefined;   // undefined
let symbolValue = Symbol("unique");  // symbol
```

JAVASCRIPT, OPERATIONS AND EXPRESSIONS

Operators:

- Operators are symbols or keywords that perform operations on operands to produce a value.
- JavaScript supports various types of operators, including:

SL.No	Operators	Definition
1	**Arithmetic**	➢ + (addition), - (subtraction), * (multiplication), / (division), % (remainder), ++ (increment), -- (decrement), etc.
2	**Comparison**	➢ == (equality), === (strict equality), != (inequality), !== (strict inequality), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), etc.
3	**Logical**	➢ && (logical AND), || (logical OR), ! (logical NOT), etc.
4	**Assignment**	➢ =, +=, -=, *=, /=, %=, etc.
5	**Bitwise**	➢ & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), << (left shift), >> (right shift), >>> (unsigned right shift), etc.
6	**Ternary**	➢ (condition ? expression1 : expression2), typeof operator, instanceof operator, etc.

Expressions:

- An expression is a combination of values, variables, operators, and function calls that evaluates to a resulting value.
- Expressions can be simple or complex, and they can include arithmetic, comparison, logical, and other operators.

For example:

1. Arithmetic expression: 3 + 4 * 2 evaluates to 11.
2. Comparison expression: 5 > 3 evaluates to true.
3. Logical expression: (x > 0) && (y < 10) evaluates to either true or false.

SCREEN OUTPUT AND KEYBOARD INPUT.

- In JavaScript, you can interact with the user through screen output and keyboard input using various methods and functions.
- Here are some common techniques for screen output and keyboard input in JavaScript:

Screen Output:

SL.No	Type	Definition
1	console.log()	➢ The console.log() method is used to display output in the browser console. It is useful for debugging and logging messages.
2	document.write()	➢ The document.write() method writes HTML content directly to the document. It is primarily used for testing and simple demonstrations.
3	DOM Manipulation	➢ You can manipulate the Document Object Model (DOM) to update the content of HTML elements dynamically.
Example	➢ document.getElementById("myElement").innerHTML = "Hello, World!";	

Keyboard Input:

SL.No	Type	Definition
1	**prompt()**	➢ The prompt() function displays a dialog box that prompts the user to enter input. The user can enter text, which is then returned as a string.
Example	**document.getElementById("myElement").innerHTML = "Hello, World!";**	
2	**Event Listeners**	➢ You can attach event listeners to specific elements or the entire document to listen for keyboard events such as key presses or key releases. This allows you to respond to user input in real-time.

Example	**document.addEventListener("keydown", function(event) {
	console.log("Key pressed: " + event.key);
	});**	

UNIT 2
ANIMATION

Animation: What is an Animation? The Start and End States, Interpolation, Animations in HTML. All About CSS Animations, Creating a Simple Animation, Detailed Look at the CSS Animation Property, Key frames, Declaring Multiple Animations, Wrap-up. All about CSS Transitions, Adding a Transition, Looking at Transitions in Detail, the Longhand Properties, Longhand Properties vs. Shorthand Properties, Working with Multiple Transitions

WHAT IS AN ANIMATION

- Animation refers to the process of creating the illusion of movement and bringing static images or objects to life.
- It involves a series of sequential images or frames that are displayed rapidly in succession, creating the perception of motion.

THE START AND END STATES

- In the context of animation, the "start state" and "end state" refer to the initial and final positions or conditions of an animated object or scene.
- They represent the beginning and ending points of an animation sequence.

Start state

- The initial state or frame from which the animation begins.
- Defines the position, appearance, and attributes of the animated elements before any movement or transformation takes place.

End state

- The final state or frame that the animated object or scene transitions to at the conclusion of the animation sequence.
- It defines the desired position, appearance, and attributes of the elements after all the intended movements, transformations, or actions have been completed.

INTERPOLATION

- Interpolation in the context of animation refers to the process of generating intermediate frames or states between two key frames to create smooth and fluid motion.
- It is a technique used to fill in the gaps between the start and end states in an animation sequence.

ANIMATIONS IN HTML

- In HTML, animations can be created using various techniques and technologies, including CSS (Cascading Style Sheets) and JavaScript.
- Here are a few methods commonly used for creating animations in HTML:

SL.No	Animations	Definition
1	**CSS Transitions**	➢ CSS transitions allow you to define the change in style or property of an element over a specified duration. ➢ Transitions are triggered by changes in CSS properties, such as hover, focus, or class toggling.
2	**CSS Keyframe Animations**	➢ CSS keyframe animations provide more complex and customizable animations. ➢ keyframes at specific percentages of an animation's duration using the @keyframes rule, you can control the intermediate states of an element.
3	**CSS Animations with Libraries**	➢ CSS animation libraries, like Animate.css or Hover.css, offer pre-built animation effects that can be easily added to HTML elements by applying predefined CSS classes. ➢ These libraries provide a wide range of animation options, including fades, slides, rotations, and more.
4	**JavaScript Animation Libraries**	➢ JavaScript libraries, such as GreenSock (GSAP) or Anime.js, provide powerful animation capabilities by leveraging JavaScript. ➢ JavaScript animations are typically triggered by events or using functions to manipulate CSS properties directly.
5	**HTML5 Canvas**	➢ For more intricate and game-like animations, the HTML5 canvas element can be utilized. ➢ The canvas provides a drawing API that allows you to create dynamic and interactive graphics using JavaScript.

ALL ABOUT CSS ANIMATIONS

- CSS animations allow you to create dynamic and visually appealing animations directly within your HTML and CSS code.
- With CSS animations, you can animate various properties of HTML elements, such as position, size, color, opacity, and more.

CREATING A SIMPLE ANIMATION

Example of creating a simple CSS animation that moves an element across the screen:

HTML:	**CSS:**

	.box {
	width: 100px;
	height: 100px;
	background-color: blue;
	position: relative;
	animation-name: slide;
	animation-duration: 2s;
	animation-timing-function: linear;
	animation-iteration-count: infinite;
	}
	@keyframes slide {
	0% {
	left: 0;
	}
	100% {
	left: 300px;
	}}

1. We create a ** ** element with the class name "box".
2. In CSS, we define the **initial styles for the "box"** element, such as its width, height, background color, and position (relative).
3. We use the **animation-name property** to specify the name of the animation, in this case, "slide".
4. The **animation-duration** property sets the duration of the animation to 2 seconds.
5. **animation-timing-function** is set to "linear" to create a constant speed motion.
6. animation-iteration-count is set to "infinite" to make the animation repeat indefinitely.
7. The **@keyframes** rule defines the intermediate states of the animation.
8. At the **beginning (0%),** the left position of the element is set to 0.
9. At the **end (100%),** the left position is set to 300px, causing the element to move 300 pixels to the right.
10. When you **view the HTML file with the provided CSS**, you'll see the blue box element moving smoothly from left to right in a continuous loop.

Detailed Look at the CSS Animation Property, Keyframes, Declaring Multiple Animations, Wrap-up

1.	**CSS Animation Property:** ➢ The animation property is a shorthand property that combines multiple animation-related properties into a single declaration.
syntax	**animation: name duration timing-function delay iteration-count direction fill-mode play-state;**
Name	Specifies the name of the keyframe animation to be applied. It can be a single animation or a comma-separated list of animations.
duration	Sets the duration of the animation, which can be specified in seconds (s) or milliseconds (ms).
timing-function	Defines the timing function that controls the acceleration and deceleration of the animation. Common values include ease, linear, ease-in, ease-out, and ease-in-out.
Delay	Specifies the delay before the animation starts, which can also be specified in seconds (s) or milliseconds (ms).
iteration-count	Sets the number of times the animation should repeat. It can be a positive integer, infinite for indefinite looping, or fractional values like 0.5 for partial iterations.
direction	Determines whether the animation plays in the forward direction (normal), backward direction (reverse), or alternates between forward and backward (alternate).
fill-mode	Specifies how the element should be styled before and after the animation. Common values include none, forwards, backwards, and both.
play-state	Allows you to pause and resume an animation using paused or running.
2.	**Keyframes:** ➢ Keyframes define the intermediate states and styles of an animation. ➢ They are specified using the @keyframes rule and divided into a series of percentage values (0% to 100%) or keywords (from and to) to represent the start and end states of the animation
Example	**@keyframes slide {** **0% {** **opacity: 0;** **}** **50% {** **opacity: 0.5;** **}** **100% {** **opacity: 1;** **}** **}**

3.	**Declaring Multiple Animations:** ➢ The animation property allows you to declare multiple animations on a single element. ➢ Simply separate the animation names with commas. ➢ Each animation can have its own duration, timing function, delay, iteration count, direction, fill mode, and play state
Example	**.element {** **animation: slide 2s ease-in-out, rotate 1.5s linear;** **}**

4. Wrap-up:

- CSS animations provide a powerful and flexible way to create dynamic and engaging visual effects on websites and web applications.
- They allow you to bring elements to life by animating properties and transitioning between states.
- Keyframes define the intermediate steps of an animation, and the animation property brings them together, controlling the animation's timing, duration, and behavior.
- By combining multiple animations and adjusting various animation properties, you can achieve more complex and intricate animations.

All about CSS Transitions

- CSS transitions allow you to create smooth and gradual changes in CSS property values over a specified duration.
- They provide a way to add animation-like effects to elements when certain properties are modified

1.	**Transition Property:** ➢ The transition property is used to specify the CSS properties that should undergo a transition effect
syntax	**transition: property duration timing-function delay;**
property	Specifies one or more CSS properties to transition. You can use shorthand properties like all, background-color, transform, etc., or specify multiple properties separated by commas.
duration	Sets the duration of the transition effect in seconds (s) or milliseconds (ms).
timing-function	Defines the timing function that controls the acceleration and deceleration of the transition. Common values include ease, linear, ease-in, ease-out, and ease-in-out.
Delay	Specifies a delay before the transition effect starts, which can also be specified in seconds (s) or milliseconds (ms).

2.	**Transition Timing:** ➢ The timing-function property within transitions determines the speed curve of the transition.
syntax	**transition: property duration timing-function delay;**
Ease	Starts slow, accelerates in the middle, and slows down at the end (default).
Linear	Progresses at a constant speed.
ease-in	Starts slow and accelerates.
ease-out	Starts fast and decelerates.
ease-in-out	Starts slow, accelerates in the middle, and decelerates at the end.

Adding a Transition

- Adding a transition to an element involves specifying the transition property and values in CSS. Here's an example of adding a transition to change the background color of a button when it's hovered over:

HTML:

```
<div class="box"></div>
```

CSS:

```
.box {
    width: 100px;
    height: 100px;
    background-color: blue;
    transition: background-color 1s;
}

.box:hover {
    background-color: red;
}
```

1. We have a ** ** element with the class "box".
2. In CSS, we set the **initial background** color of the button to blue, along with other styles like text color and padding.
3. The **transition property is applied** to the box, specifying that the background-color property should transition with a duration of 1s seconds and an easing function of ease.
4. When the **button is hovered over** (using the :hover pseudo-class), the background color is changed to red.
5. When you **view the HTML file with the provided CSS**, you'll see that the button smoothly transitions from blue to red when hovered over.

Transition longhand:

- The option to use longhand properties to individually control different aspects of the transition.

Example	**.element {** **Transition-property : color;** **Transition-delay : 500ms;** **Transition-duration : 2s;** **Transition-time-function : ease-in;** **}**

Transition Shorthand:

- Shorthand properties provide a concise way to define multiple aspects of the transition in a single declaration.

Example	**.element {** **Transition: color 2s ease-in 500ms** **}**

Longhand Properties vs. Shorthand Properties

Longhand Properties:	**Shorthand Properties:**
➢ Longhand properties allow you to control specific aspects of the transition-related property, declaration. duration, timing function, and delay. ➢ Each property is declared separately using its respective CSS property name. ➢ Provides more fine-grained control over the transition effects. ➢ Offers flexibility to specify different values each property.	➢ Shorthand properties combine multiple individually related properties into a single transition, such as ➢ Simplify the code by condensing multiple properties into one. ➢ Provide a more concise and readable way to define transitions. ➢ Useful when you want to apply the same values to multiple properties simultaneously. for
.element { **transition-property: width;** **transition-duration: 1s;** **transition-timing-function: ease-in-out;** **transition-delay: 0.3s;** **}**	**.element {** **transition: width 1s ease-in-out 0.3s;** **}**
Longhand properties offer more control and flexibility when you need different values for to aspect of the transition effect.	Shorthand properties are handy when you want apply the same transition settings to multiple each properties, making the code shorter and more readable.

Working with Multiple Transitions.

- apply different transition effects to individual CSS properties or groups of properties

1. **Applying Different Transitions to Individual Properties:**

- Use the transition or longhand properties on separate lines to define differenttransitions for different properties.
- Each line specifies the transition effect for a specific CSS property.

Example

```
.element {
  transition: width 1s ease-in-out;
  transition-property: height;
  transition-duration: 0.5s;
}
```

2. **Grouping Multiple Properties in a Single Transition:**

- If multiple properties should have the same transition effect, you can group themtogether within a single transition or longhand declaration.
- Separate the property names with commas to define the grouped properties

Example

```
.element {
  transition: width 1s ease-in-out, height 0.5s linear;
}
```

3. **Transitioning All Properties:**

- To apply the same transition effect to all eligible properties, you can use the all keyword in the transition or transition-property declaration.

Example

```
.element {
  transition: all 1s ease-in-out;
}
```

UNIT 3
HTML5-SVG

HTML5 – SVG: Viewing SVG Files, Embedding SVG in HTML5, HTML5 − SVG Circle, HTML5 − SVG Rectangle, HTML5 − SVG Line, HTML5 − SVG Ellipse, HTML5 − SVG Polygon, HTML5 − SVG Polyline, HTML5 − SVG Gradients, HTML5 − SVG Star.

HTML5 SVG

- SVG(Scalable Vector Graphics) is a markup language used to create vector-based graphics and animations on the web.
- It allows for the creation of scalable and resolution-independent images that can be manipulated and animated using JavaScript or CSS.
- SVG is supported by all modern web browsers and provides a wide range of features and elements to create visually appealing and interactive graphics

Key features of HTML5 SVG include:

Sl.No	Feature	Definition
1	Basic Shapes	Basic Shapes: SVG provides elements such as <circle>, <rect>, <line>, <ellipse>, <polygon>, and <polyline> to create basic shapes on the canvas.
2	Paths	SVG allows you to create complex shapes using path commands like <path> element, which supports various commands like M (move to), L (line to), C (cubic Bezier curve), Q (quadratic Bezier curve), and more.
3	Text	SVG supports the <text> element to display text within the graphic, allowing you to control the font, size, alignment, and other text-related properties.
4	Transformations	SVG provides transformations like translation, rotation, scaling, and skewing through attributes like transform, allowing you to manipulate and animate elements in the SVG canvas.
5	Gradients	SVG supports linear and radial gradients using the <linearGradient> and <radialGradient> elements, allowing you to create smooth color transitions across shapes and objects.
6	Filters and Effects	SVG offers a variety of filters and effects to apply to shapes and images, such as blurring, drop shadows, color adjustments, and more, using the <filter> element.
7	Animation	SVG supports animation and interactivity through CSS animations, JavaScript, and the <animate> element. You can animate properties like position, size, color, opacity, and more to create dynamic and engaging visuals.
8	Integration with HTML5	SVG can be seamlessly integrated with HTML5 elements and other technologies. You can embed SVG directly within an HTML document using the <svg> element or include it as an external file using the <object> or <img> element.

Viewing SVG Files

- To view SVG files, you have a few options:

1	**Web Browsers**	➢ Most modern web browsers, such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge, have built-in support for rendering SVG files. ➢ You can simply open an SVG file in your web browser by selecting "File" > "Open File" or by dragging and dropping the SVG file into the browser window.
2	**Online SVG Viewers**	➢ There are various online tools and websites that allow you to upload and view SVG files directly in your browser. ➢ Some popular options include SVG Viewer by SVG-Edit (https://svg-edit.github.io/svgedit/dist/svg-editor.html) and SVG Viewer by Codepen (https://codepen.io/SVGViewer/details/bdVXbO).
3	**SVG Editing Software**	➢ SVG files can be opened and viewed using dedicated SVG editing software. ➢ Some popular options include Adobe Illustrator, Inkscape, Sketch, and CorelDRAW. ➢ These tools not only allow you to view SVG files but also provide advanced editing capabilities for modifying and creating SVG graphics.
4	**Text Editors**	➢ Since SVG files are XML-based, you can open them in any text editor, such as Notepad, Sublime Text, Visual Studio Code, or Atom. ➢ While this option doesn't provide a visual representation of the SVG, it allows you to inspect and modify the code directly.

Embedding SVG in HTML5

- To embed an SVG (Scalable Vector Graphics) file in an HTML5 document, you can use the <svg> element.

Here's an example of how to embed an SVG file directly within an HTML document:

```
<!DOCTYPE html>
<html>
<head>
  <title>Embedding SVG in HTML5</title>
</head>
<body>
  <h2>Embedded SVG Example</h2>
  <svg width="400" height="300">
    <image xlink:href="image.svg" width="400" height="300" />
  </svg>
</body>
</html>
```

In the above example:

1. The <svg> element is used to define a container for the SVG content.
2. The width and height attributes specify the dimensions of the SVG canvas.
3. The <image> element is used to embed the SVG file. The xlink:href attribute specifies the path to the SVG file. Adjust the width and height attributes of the <image> element to match the desired dimensions of the SVG image within the container.

Make sure to replace "image.svg" with the actual path to your SVG file.

Save this code to an HTML file and open it in a web browser.
The SVG image specified in the <image> element will be displayed within the SVG container with the specified dimensions.

1. HTML5– SVG CIRCLE

- To create an SVG circle in HTML5, you can use the <circle> element.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>SVG Circle</title>
</head>
<body>
  <svg width="400" height="400">
    <circle cx="200" cy="200" r="100" fill="red" />
  </svg>
</body>
</html>
```

<svg>	The <svg> element creates the SVG container with a width and height of 400 pixels.
<circle>	The <circle> element represents the circle shape.
cx	The cx attribute defines the x-coordinate of the center of the circle (200 in this case).
cy	The cy attribute defines the y-coordinate of the center of the circle (200 in this case).
r	The r attribute specifies the radius of the circle (100 in this case).
fill	The fill attribute sets the fill color of the circle (red in this

2. HTML5 − SVG Rectangle

- To create an SVG rectangle in HTML5, you can use the <rect> element.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>SVG Rectangle</title>
</head>
<body>
  <svg width="400" height="400">
    <rect x="50" y="100" width="300" height="200" fill="blue" />
  </svg>
</body>
</html>
```

<svg>	The <svg> element creates the SVG container with a width and height of 400 pixels.
<rect>	The <rect> element represents the rectangle shape.
x	The x attribute defines the x-coordinate of the top-left corner of the rectangle (50 in this case).
y	The y attribute defines the y-coordinate of the top-left corner of the rectangle (100 in this case).
width	The width attribute specifies the width of the rectangle (300 in this case).
height	The height attribute specifies the height of the rectangle (200 in this case).
fill	The fill attribute sets the fill color of the rectangle (blue in this case).

3. HTML5 − SVG Line

- To create an SVG line in HTML5, you can use the <line> element.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>SVG Line</title>
</head>
<body>
  <svg width="400" height="400">
    <line x1="50" y1="100" x2="350" y2="300" stroke="green" stroke-
width="2" />
```

<line>	The <line> element represents the line shape.
x1	The x1 attribute defines the x-coordinate of the starting point of the line (50 in this case).
y1	The y1 attribute defines the y-coordinate of the starting point of the line (100 in this case).
x2	The x2 attribute defines the x-coordinate of the ending point of the line (350 in this case).
y2	The y2 attribute defines the y-coordinate of the ending point of the line (300 in this case).
stroke	The stroke attribute sets the color of the line (green in this case).
stroke-Width	The stroke-width attribute specifies the thickness of the line (2 in this case).

4. HTML5 − SVG Ellipse

- To create an SVG ellipse in HTML5, you can use the <ellipse> element.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>SVG Ellipse</title>
</head>
<body>
  <svg width="400" height="400">
    <ellipse cx="200" cy="200" rx="150" ry="100" fill="yellow" />
  </svg>
</body>
</html>
```

<svg>	The <svg> element creates the SVG container with a width and height of 400 pixels.
<ellipse>	The <ellipse> element represents the ellipse shape.
cx	The cx attribute defines the x-coordinate of the center of the ellipse (200 in this case).
cy	The cy attribute defines the y-coordinate of the center of the ellipse (200 in this case).
rx	The rx attribute specifies the horizontal radius of the ellipse (150 in this case).
ry	The ry attribute specifies the vertical radius of the ellipse (100 in this case).
fill	The fill attribute sets the fill color of the ellipse (yellow in this case).

5. HTML5– SVG Polygon

- To create an SVG polygon in HTML5, you can use the <polygon> element.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>SVG Polygon</title>
</head>  <body>
  <svg width="400" height="400">
    <polygon points="200,50 350,200 200,350 50,200" fill="orange" />
  </svg>
</body>  </html>
```

<svg>	The <svg> element creates the SVG container with a width and height of 400 pixels.
<polygon>	The <polygon> element represents the polygon shape.
points	The points attribute defines the coordinates of the vertices of the polygon. In this case, the polygon has four vertices: (200, 50), (350, 200), (200, 350), and (50, 200). The coordinates are specified in pairs separated by spaces.
fill	The fill attribute sets the fill color of the ellipse (orange in this case).

6. HTML5– Polyline

- To create an SVG polyline in HTML5, you can use the <polyline> element.

Example

```
<!DOCTYPE html>
<html>
<head> <title>SVG Polyline</title>
</head>  <body>
  <svg width="400" height="400">
    <polyline points="100,200 200,50 300,200 200,350" fill="none" stroke="purple" stroke-width="2" />
  </svg>  </body>  </html>
```

<svg>	The <svg> element creates the SVG container with a width and height of 400 pixels.
<polyline>	The <polyline> element represents the polyline shape.
points	The points attribute defines the coordinates of the vertices of the polyline. In this case, the polyline has four vertices: (100, 200), (200, 50), (300, 200), and (200, 350). The coordinates are specified in pairs separated by spaces.
fill	The fill="none" attribute specifies that the polyline should have no fill color.
stroke	The stroke="purple" attribute sets the stroke color of the polyline (purple in this case).
stroke-width	The stroke-width="2" attribute specifies the thickness of the polyline (2 in this case).

6. HTML5 − SVG Gradients

To create SVG gradients in HTML5, you can use the <linearGradient> and <radialGradient> elements.

Example

```
<!DOCTYPE html>
<html>
<head>
   <title>SVG Gradients</title>
</head>
<body>
   <svg width="400" height="200">
      <defs>
         <linearGradient id="grad1" x1="0%" y1="0%" x2="100%" y2="0%">
            <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
            <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
         </linearGradient>
         <radialGradient id="grad2" cx="50%" cy="50%" r="50%" fx="50%"
fy="50%">
            <stop offset="0%" style="stop-color:rgb(255,255,255);stop-opacity:1" />
            <stop offset="100%" style="stop-color:rgb(0,0,255);stop-opacity:1" />
         </radialGradient>
      </defs>
      <rect x="50" y="50" width="300" height="100" fill="url(#grad1)" />
      <circle cx="200" cy="150" r="70" fill="url(#grad2)" />
   </svg>
</body> </html>
```

<svg>	The <svg> element creates the SVG container with a width of 400 pixels and a height of 200 pixels.
<defs>	The <defs> element is used to define the gradients.
<linearGradient>	The <linearGradient> element with id="grad1" defines a linear gradient that starts with a yellow color at the left (x1="0%" y1="0%") and ends with a red color at the right (x2="100%" y2="0%").
<stop>	Inside the <linearGradient>, there are two <stop> elements that define the color stops of the gradient. The offset attribute specifies the position along the gradient, and the style attribute sets the color and opacity.
<radialGradient>	The <radialGradient> element with id="grad2" defines a radial gradient that starts with a white color at the center (cx="50%" cy="50%") and ends with a blue color at the edges (fx="50%" fy="50%").
<svg>	Inside the <svg> element, a <rect> and a <circle> are created. The fill attribute of each shape is set to "url(#grad1)" and "url(#grad2)", respectively, to apply the gradients.

7. HTML5 − SVG Star

- To create an SVG star in HTML5, you can use the <polygon> element.

Example

```
<!DOCTYPE html>
<html>
<head>
  <title>SVG Star</title>
</head>
<body>
  <svg width="400" height="200">
    <polygon points="200,10 250,190 60,70 340,70 150,190" fill="gold" />
  </svg>
</body>
</html>
```

<svg>	The <svg> element creates the SVG container with a width of 400 pixels and a height of 200 pixels.
<polygon>	The <polygon> element represents the star shape.
points	The points attribute defines the coordinates of the vertices of the star. In this case, the star has five vertices: (200, 10), (250, 190), (60, 70), (340, 70), and (150, 190). The coordinates are specified in pairs separated by spaces.
fill	The fill attribute sets the fill color of the star (gold in this case).

UNIT 4
HTML5 – CANVAS

HTML5 – CANVAS: The Rendering Context, Browser Support, HTML5 Canvas Examples, Canvas - Drawing Rectangles, Canvas - Drawing Paths, Canvas - Drawing Lines, Canvas - Drawing Bezier Curves, Canvas - Drawing Quadratic Curves, Canvas - Using Images, Canvas - Create Gradients.

HTML5 – CANVAS

- HTML5 Canvas is a powerful feature introduced in HTML5 that allows you to dynamically render graphics, animations, and images directly in a web page using JavaScript.
- With the Canvas API, you have full control over individual pixels and can create complex visualizations, games, data visualizations, and more.
- To use the HTML5 Canvas, you need to add a <canvas> element to your HTML markup.
- This element acts as a drawing surface and can be styled using CSS**.**

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Canvas Example</title>
    <style>
      #myCanvas {
        border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <canvas id="myCanvas" width="500" height="500"></canvas>
       <script>
      // JavaScript code to draw on the canvas goes here
    </script>
  </body>
</html>
```

- We have created a canvas element with an id of "myCanvas" and specified its width and height as 500 pixels.
- We have also added a simple border using CSS.
- To draw on the canvas, you need to access the canvas element using JavaScript and obtain its 2D rendering context.

Example that draws a blue rectangle on the canvas:

```
<script>
  var canvas = document.getElementById('myCanvas');
  var ctx = canvas.getContext('2d');

  ctx.fillStyle = 'blue';
  ctx.fillRect(50, 50, 100, 100);
</script>
```

The Rendering Context

- The rendering context in HTML5 Canvas is an object that provides methods and properties for drawing and manipulating graphics on the canvas.
- The rendering context is obtained using the getContext method of the canvas element, and it determines the type of graphics that can be rendered on the canvas.

There are two main types of rendering contexts:

1. 2D Rendering Context:

- The 2D rendering context, accessed by using the string '2d' as the argument to get Context, provides methods for drawing and manipulating 2D graphics.
- This is the most commonly used rendering context in HTML5 Canvas.

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
```

2. WebGL Rendering Context:

- The WebGL rendering context, accessed by using the string 'webgl' or 'experimental-webgl' as the argument to getContext, provides a 3D graphics rendering API based on OpenGL ES.
- WebGL allows you to create interactive 3D graphics and games directly in the browser using JavaScript.

Example

```
var canvas = document.getElementById('myCanvas');
var gl = canvas.getContext('webgl');
```

NOTE

- The WebGL rendering context exposes a low-level API for working with shaders, buffers, textures, and other WebGL-specific concepts.
- It requires a good understanding of computer graphics and shader programming to use effectively.
- Each rendering context has its own set of methods and properties that are specific to its capabilities.

BROWSER SUPPORT

- HTML5 Canvas has excellent browser support and is widely supported across modern web browsers.
- It is supported in major desktop browsers, mobile browsers, and even in some older versions of browsers.
- Here is an overview of the browser support for HTML5 Canvas as of my knowledge cutoff in September 2021:

1. Chrome: Full support in all versions.
2. Firefox: Full support in all versions.
3. Safari: Full support in all versions.
4. Edge: Full support in all versions.
5. Opera: Full support in all versions.
6. Internet Explorer: Partial support starting from Internet Explorer 9.
   However, older versions of Internet Explorer (8 and below) do not support Canvas.

HTML5 Canvas Examples

1. Drawing Shapes
2. Creating Animations
3. Image Manipulation
4. Interactive Drawing

1. Drawing Shapes:

- Canvas to draw various shapes such as rectangles, circles, lines, and polygons.
- Here's an example of drawing a rectangle and a circle:

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
// Draw a rectangle
ctx.fillStyle = 'blue';
ctx.fillRect(50, 50, 100, 100);
// Draw a circle
ctx.beginPath();
ctx.arc(200, 200, 50, 0, 2 * Math.PI);
ctx.fillStyle = 'red';
ctx.fill();
ctx.closePath();
```

2. Creating Animations:

- HTML5 Canvas is great for creating animations.
- By redrawing the canvas at regular intervals, you can create dynamic and interactive animations.
- Here's an example of a bouncing ball animation:

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
var x = canvas.width / 2;
var y = canvas.height / 2;
var dx = 2;
var dy = -2;
var ballRadius = 20;
function drawBall() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.beginPath();
  ctx.arc(x, y, ballRadius, 0, 2 * Math.PI);
  ctx.fillStyle = 'blue';
  ctx.fill();
  ctx.closePath();
  x += dx;
  y += dy;
  requestAnimationFrame(drawBall);
}
drawBall();
```

3. Image Manipulation:

- **load and manipulate images on the Canvas.**
- **Here's an example of how to draw an image on the Canvas:**

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

var img = new Image();
img.src = 'image.jpg';

img.onload = function() {
  ctx.drawImage(img, 0, 0);
};
```

3. Interactive Drawing:

- **Allow users to draw on the Canvas using mouse or touch events.**
- **Here's an example of a simple interactive drawing application:**

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
var isDrawing = false;
canvas.addEventListener('mousedown', startDrawing);
canvas.addEventListener('mousemove', draw);
canvas.addEventListener('mouseup', stopDrawing);
canvas.addEventListener('mouseout', stopDrawing);
function startDrawing(e) {
  isDrawing = true;
  ctx.beginPath();
  ctx.moveTo(e.clientX, e.clientY);
}
function draw(e) {
  if (!isDrawing) return;
  ctx.lineTo(e.clientX, e.clientY);
  ctx.stroke();
}
function stopDrawing() {
  isDrawing = false;
}
```

1. Canvas - Drawing Rectangles

- Drawing rectangles on the HTML5 Canvas is a common task.
- You can use the fillRect and strokeRect methods of the 2D rendering context to draw filled or outlined rectangles, respectively.
- Here's an example that demonstrates drawing rectangles on the canvas

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
// Drawing a filled rectangle
ctx.fillStyle = 'blue';
ctx.fillRect(50, 50, 200, 100);
// Drawing an outlined rectangle
ctx.strokeStyle = 'red';
ctx.lineWidth = 2;
ctx.strokeRect(100, 150, 150, 80);
```

Canvas - Drawing Rectangles (above code)

1. We first obtain the 2D rendering context from the canvas element.
2. We set the fill style using the fillStyle property to specify the color of the filled rectangle (blue in this case).
3. Then we call the fillRect method, providing the position (x, y) and dimensions (width, height) of the rectangle.
4. Next, we set the stroke style using the strokeStyle property to specify the color of the outline (red in this case).
5. We can also set the line width using the lineWidth property to control the thickness of the outline.
6. Finally, we call the strokeRect method, providing the position and dimensions of the outlined rectangle.
7. You can customize the position, size, colors, and other properties of the rectangles to achieve the desired effect.
8. By using these methods along with other drawing and styling functions, you can create complex shapes and visual elements on the canvas.

2. Canvas - Drawing Paths

- Drawing paths on the HTML5 Canvas allows you to create custom shapes and lines by defining a series of points and connecting them with lines or curves.
- The Canvas API provides methods like beginPath, moveTo, lineTo, arc, and bezierCurveTo to create paths.
- Here's an example that demonstrates drawing paths on the canvas:

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
// Drawing a triangle
ctx.beginPath();
ctx.moveTo(100, 100); // Starting point
ctx.lineTo(200, 100); // Line to point (200, 100)
ctx.lineTo(150, 200); // Line to point (150, 200)
ctx.closePath(); // Connects the last point to the starting point
ctx.fillStyle = 'blue';
ctx.fill();
// Drawing a circle
ctx.beginPath();
ctx.arc(300, 150, 50, 0, 2 * Math.PI); // Circle centered at (300, 150) with radius 50
ctx.fillStyle = 'red';
ctx.fill();
// Drawing a curved path
ctx.beginPath();
ctx.moveTo(400, 100); // Starting point
```

```
ctx.quadraticCurveTo(450, 200, 500, 100); // Quadratic curve from (450, 200) to (500, 100)
ctx.strokeStyle = 'green';
ctx.lineWidth = 3;
ctx.stroke();
```

Canvas - Drawing Paths (above code)

1. We first obtain the 2D rendering context from the canvas element.
2. Then, we use the beginPath method to start a new path.
3. We can then define the path by calling different methods:
4. moveTo sets the starting point of the path.
5. lineTo connects the current point to the specified point with a straight line.
6. arc creates an arc or circle.
7. bezierCurveTo creates a curve using cubic Bezier control points.
8. quadraticCurveTo creates a curve using quadratic Bezier control points.
9. After defining the path, we can style and render it using fill or stroke operations. fill fills the path with a specified color, while stroke outlines the path.
10. You can set the fill and stroke styles using the fillStyle and strokeStyle properties.
11. Remember to use beginPath before starting a new path to ensure that the previous path is not connected to the new one unintentionally.
12. By combining different path creation methods, you can create complex shapes, curves, and lines on the canvas.

4. Canvas - Drawing Lines

- Drawing lines on the HTML5 Canvas is straightforward using the moveTo and lineTo methods of the 2D rendering context.
- These methods allow you to specify the starting point and subsequent points to draw a connected line.
- Here's an example that demonstrates drawing lines on the canvas:

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
// Drawing a straight line
ctx.beginPath();
ctx.moveTo(50, 50); // Starting point
ctx.lineTo(200, 50); // Line to point (200, 50)
ctx.strokeStyle = 'blue';
ctx.lineWidth = 2;
ctx.stroke();
// Drawing a diagonal line
ctx.beginPath();
ctx.moveTo(50, 100); // Starting point
ctx.lineTo(200, 200); // Line to point (200, 200)
ctx.strokeStyle = 'red';
```

```
ctx.lineWidth = 3;
ctx.stroke();
```

Canvas - Drawing Lines (above code)

1. We first obtain the 2D rendering context from the canvas element.
2. Then, for each line we want to draw, we start a new path using beginPath.
3. We set the starting point of the line using moveTo, specifying the initial coordinates.
4. We then use lineTo to draw a line from the starting point to the desired end point.
5. After defining the line, we can set the stroke style using the strokeStyle property to specify the color of the line.
6. The lineWidth property allows us to set the thickness of the line.
7. Finally, we call stroke to render the line on the canvas.
8. You can repeat these steps to draw multiple lines with different positions, colors, and thicknesses.
9. Remember to call beginPath before starting a new line to ensure that the previous line is not connected to the new one unintentionally.
10. By combining the moveTo and lineTo methods, you can create complex line patterns, shapes, and even curves on the canvas.

5. Canvas - Drawing Bezier Curves

- Drawing Bezier curves on the HTML5 Canvas allows you to create smooth and curved paths.
- The Canvas API provides methods such as quadraticCurveTo and bezierCurveTo to define and render Bezier curves.
- Here's an example that demonstrates drawing Bezier curves on the canvas:

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
// Drawing a quadratic Bezier curve
ctx.beginPath();
ctx.moveTo(50, 150); // Starting point
ctx.quadraticCurveTo(100, 50, 200, 150); // Control point (100, 50) and end point (200, 150)
ctx.strokeStyle = 'blue';
ctx.lineWidth = 2;
ctx.stroke();
// Drawing a cubic Bezier curve
ctx.beginPath();
ctx.moveTo(250, 150); // Starting point
ctx.bezierCurveTo(300, 50, 400, 250, 450, 150); // Control points (300, 50) and (400, 250), and end point (450, 150)
ctx.strokeStyle = 'red';
ctx.lineWidth = 2;
ctx.stroke();
```

Canvas - Drawing Bezier Curves (above code)

1. we first obtain the 2D rendering context from the canvas element.
2. Then, for each Bezier curve we want to draw, we start a new path using beginPath.
3. We set the starting point of the curve using moveTo, specifying the initial coordinates.
4. For quadratic Bezier curves, we use quadraticCurveTo to define the curve.
5. It takes two arguments: the control point's coordinates and the end point's coordinates.
6. The control point influences the shape of the curve.
7. For cubic Bezier curves, we use bezierCurveTo to define the curve.
8. It takes three arguments: two control points' coordinates and the end point's coordinates.
9. The control points allow for more control over the shape of the curve.
10. After defining the curve, we can set the stroke style using the strokeStyle property to specify the color of the curve.
11. The lineWidth property allows us to set the thickness of the curve. Finally, we call stroke to render the curve on the canvas.
12. You can experiment with different control point and end point coordinates to create a variety of curved shapes and lines on the canvas.
13. Remember to call beginPath before starting a new curve to ensure that the previous curve is not connected to the new one unintentionally.
14. By using Bezier curves, you can achieve smooth and complex curves for creating illustrations, animations, or other graphical elements on the canvas.

6. Canvas - Drawing Quadratic Curves

- Drawing quadratic curves on the HTML5 Canvas allows you to create smooth and curved paths using control points.
- The Canvas API provides the quadraticCurveTo method to define and render quadratic curves.
- Here's an example that demonstrates drawing quadratic curves on the canvas:

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
// Drawing a quadratic curve
ctx.beginPath();
ctx.moveTo(50, 150); // Starting point
ctx.quadraticCurveTo(100, 50, 200, 150); // Control point (100, 50) and end point (200, 150)
ctx.strokeStyle = 'blue';
ctx.lineWidth = 2;
ctx.stroke();
```

Canvas - Drawing Quadratic Curves(above code)

1. We first obtain the 2D rendering context from the canvas element.
2. Then, we start a new path using beginPath.
3. We set the starting point of the curve using moveTo, specifying the initial coordinates.

4. To define the quadratic curve, we use quadraticCurveTo method, which takes two arguments: the control point's coordinates and the end point's coordinates.
5. The control point influences the shape and direction of the curve.
6. After defining the curve, we can set the stroke style using the strokeStyle property to specify the color of the curve.
7. The lineWidth property allows us to set the thickness of the curve.
8. Finally, we call stroke to render the curve on the canvas.
9. You can experiment with different control point and end point coordinates to create various quadratic curves and shapes on the canvas.
10. Remember to call beginPath before starting a new curve to ensure that the previous curve is not connected to the new one unintentionally.
11. By utilizing quadratic curves, you can achieve smooth and rounded shapes, lines, and animations on the canvas.

7. Canvas - Using Images

- Using images on the HTML5 Canvas allows you to display and manipulate image content within your canvas-based applications.
- The Canvas API provides the drawImage method to load and draw images onto the canvas.
- Here's an example that demonstrates how to use images on the canvas:

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
var img = new Image();
img.src = 'image.jpg';
img.onload = function() {
ctx.drawImage(img, 0, 0);
};
```

Canvas - Using Images (above code)

1. We first obtain the 2D rendering context from the canvas element.
2. We then create a new Image object and assign it the source URL of the image file using the src property.
3. Make sure to replace 'image.jpg' with the actual path to your image file.
4. We set up an onload event handler for the image, which ensures that the image is fully loaded before attempting to draw it on the canvas.
5. Inside the onload function, we use the drawImage method to draw the image onto the canvas.
6. The first argument is the image object, and the second and third arguments specify the starting position (x, y) on the canvas where the image should be drawn.
7. You can customize the position, size, and other properties of the drawn image by providing additional arguments to the drawImage method.
8. For example, you can specify the destination width and height to scale the image, or use the source and destination rectangles to crop and resize the image.

9. Remember that the image must be hosted on the same domain as the web page to avoid security restrictions due to cross-origin resource sharing (CORS).
10. By incorporating image loading and drawing into your canvas applications, you can create dynamic visualizations, games, and interactive experiences that incorporate image assets.

8. Canvas - Create Gradients

- Creating gradients on the HTML5 Canvas allows you to fill shapes and regions with smooth color transitions.
- The Canvas API provides two types of gradients: linear gradients and radial gradients.
- You can use the createLinearGradient and createRadialGradient methods of the 2D rendering context to create gradients, and then use them to fill shapes using the fill method.
- Here's an example that demonstrates creating and using gradients on the canvas:

Example

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');

// Creating a linear gradient
var linearGradient = ctx.createLinearGradient(0, 0, canvas.width, 0);
linearGradient.addColorStop(0, 'red');
linearGradient.addColorStop(0.5, 'green');
linearGradient.addColorStop(1, 'blue');

// Drawing a rectangle filled with the linear gradient
ctx.fillStyle = linearGradient;
ctx.fillRect(50, 50, 200, 100);

// Creating a radial gradient
var radialGradient = ctx.createRadialGradient(300, 200, 50, 300, 200, 150);
radialGradient.addColorStop(0, 'yellow');
radialGradient.addColorStop(1, 'orange');

// Drawing a circle filled with the radial gradient
ctx.fillStyle = radialGradient;
ctx.beginPath();
ctx.arc(300, 200, 150, 0, 2 * Math.PI);
ctx.fill();
```

Canvas - Using Images (above code)

1. We first obtain the 2D rendering context from the canvas element.
2. Then, we create a linear gradient using the createLinearGradient method, specifying the starting and ending points of the gradient as (x0, y0, x1, y1).
3. We then use the addColorStop method to define color stops along the gradient.
4. Each color stop consists of a position (ranging from 0 to 1) and a color.

5. After creating the gradient, we set the fillStyle property of the context to the gradient and draw a rectangle using the fillRect method.
6. The rectangle will be filled with the linear gradient.
7. Next, we create a radial gradient using the createRadialGradient method, specifying the center and radius of the inner and outer circles of the gradient.
8. Again, we use addColorStop to define color stops along the gradient.
9. Finally, we set the fillStyle property to the radial gradient and draw a circle using the arc and fill methods.
10. The circle will be filled with the radial gradient.
11. You can customize the positions, colors, and number of color stops to create various gradient effects on the canvas.
12. By leveraging gradients, you can enhance the visual appeal of your canvas-based applications and create smooth color transitions in your drawings and shapes.

UNIT 5
HTML5 - STYLES AND COLORS

HTML5 - Styles and Colors, Canvas - Text and Fonts, Canvas - Pattern and Shadow, Canvas - Save and Restore States, Canvas - Translation, Canvas - Rotation, Canvas - Scaling, Canvas - Transforms, HTML5 Canvas - Composition, Canvas – Animations..

HTML5 - Styles and Colors

- Styles and colors to elements using CSS (Cascading Style Sheets).
- CSS provides a wide range of options for customizing the appearance of HTML elements.

Styles and colors in HTML5:

Sl.No	Styles and colors	Example
1	**Inline Styles**	You can apply styles directly to an HTML element using the style attribute. **For example:** Hello, World!

2	**Internal Stylesheets**	You can define styles within the <style> tags in the <head> section of your HTML document.
		For example:
		<head>
		<style>
		h1 {
		color: red;
		font-size: 24px;
		}
		</style>
		</head>
		<body>
		 Welcome!
		</body>

3	**External Stylesheets**	You can also define styles in an external CSS file and link it to your HTML document using the <link> tag. **For example:** <head> <link rel="stylesheet" href="styles.css"> </head> <body> Welcome! </body>

4	**Color**	CSS provides various ways to specify colors, including named colors, hexadecimal values, RGB values, and HSL values.
		For example:
		h1 {
		color: red; /* Named color */
		background-color: #00ff00; /* Hexadecimal color */
		border-color: rgb(255, 0, 0); /* RGB color */
		box-shadow: hsl(240, 100%, 50%) 2px 2px 2px; /* HSL color */
		}

1. Canvas - Text and Fonts

- In HTML5 Canvas, you can draw text on the canvas and customize the font properties.
- You can use the fillText() or strokeText() methods to render text, and the font property to control the font style.
- Here's an example of drawing text on a canvas:

Example

```
// Get the canvas element
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");

// Set the font properties
ctx.font = "24px Arial";
ctx.fillStyle = "red";
ctx.textAlign = "center";

// Draw filled text
ctx.fillText("Hello, World!", canvas.width/2, canvas.height/2);

// Draw stroked text
ctx.strokeStyle = "blue";
ctx.lineWidth = 2;
ctx.strokeText("Hello, World!", canvas.width/2, canvas.height/2 + 50);
```

In the example above:

1. We obtain a reference to the canvas element and its 2D rendering context.
2. We set the font property to specify the font size and family.
3. In this case, the font size is set to 24 pixels and the font family is set to Arial.
4. We set the fillStyle property to determine the color of the filled text.
5. We set the textAlign property to specify the alignment of the text.
6. In this case, it is set to "center" to center the text horizontally.

7. We use the fillText() method to draw filled text on the canvas.
8. It takes the text to be drawn, followed by the x and y coordinates of the baseline position.
9. We set the strokeStyle property to determine the color of the text outline when using strokeText().
10. We set the lineWidth property to specify the thickness of the text outline.
11. We use the strokeText() method to draw the outlined text on the canvas, using the same parameters as fillText().

2. Canvas - Pattern

- Patterns in Canvas allow you to fill shapes with repeating images or custom patterns.
- You can create a pattern using the createPattern() method, which takes an image or another canvas as the pattern source.
- Here's an example:

Example

```
// Get the canvas element
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");

// Create a pattern from an image
var img = new Image();
img.src = "pattern-image.png";
img.onload = function() {
  var pattern = ctx.createPattern(img, "repeat");
  ctx.fillStyle = pattern;
  ctx.fillRect(0, 0, canvas.width, canvas.height);
};
```

In the above example:

1. We obtain a reference to the canvas element and its 2D rendering context.
2. We create an Image object and set its source to the pattern image file.
3. After the image is loaded, we use the createPattern() method to create a pattern from the image.
4. The second argument specifies how the pattern should repeat.
5. Options include "repeat" (default), "repeat-x", "repeat-y", or "no-repeat".
6. We set the fillStyle to the created pattern.
7. Finally, we draw a rectangle that fills the entire canvas using fillRect().

3. Canvas Shadows:

- Shadows in Canvas allow you to add a visual shadow effect to your drawings.
- You can set the shadow properties using the shadowOffsetX, shadowOffsetY, shadowBlur, and shadowColor properties.
- Here's an example:

Example

```
// Get the canvas element
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");

// Set shadow properties
ctx.shadowOffsetX = 4;
ctx.shadowOffsetY = 4;
ctx.shadowBlur = 6;
ctx.shadowColor = "rgba(0, 0, 0, 0.5)";

// Draw a rectangle with a shadow
ctx.fillStyle = "red";
ctx.fillRect(50, 50, 200, 100);
```

In the above example:

1. We obtain a reference to the canvas element and its 2D rendering context.
2. We set the shadowOffsetX and shadowOffsetY properties to define the horizontal and vertical offsets of the shadow from the object.
3. We set the shadowBlur property to specify the blurriness of the shadow.
4. We set the shadowColor property to determine the color and transparency of the shadow.
5. Finally, we draw a rectangle with a shadow using fillRect().
6. By applying patterns and shadows, you can add depth, texture, and visual interest to your canvas drawings.
7. Experiment with different patterns, images, shadow properties, and drawing techniques to achieve the desired effects.

4. Canvas - Save and Restore States

- In HTML5 Canvas, you can save and restore the drawing state using the save() and restore() methods.
- These methods allow you to store and retrieve the current state of the canvas, including transformations, styles, and other settings.
- Here's an example that demonstrates how to use save() and restore():

Example

```
// Get the canvas element
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");

// Save the current state
ctx.save();

// Perform transformations and style changes
```

```
ctx.fillStyle = "red";
ctx.translate(100, 100);
ctx.rotate(Math.PI / 4);

// Draw a rectangle with the modified state
ctx.fillRect(0, 0, 100, 50);

// Restore the previous state
ctx.restore();

// Draw another rectangle with the restored state
ctx.fillRect(200, 200, 100, 50);
```

In the example above:

1. We obtain a reference to the canvas element and its 2D rendering context.
2. We use the save() method to save the current state of the canvas.
3. This includes transformations (such as translations and rotations) and style changes (such as fill colors and line widths).
4. After saving the state, we perform transformations and style changes as needed.
5. We draw a rectangle using fillRect() with the modified state.
6. This rectangle will have the applied transformations and styles.
7. We use the restore() method to restore the previously saved state.
8. This reverts the transformations and style changes made since the last save() call.
9. Finally, we draw another rectangle using fillRect() with the restored state.
10. This rectangle will be drawn with the original transformations and styles.
11. By using save() and restore(), you can isolate specific modifications to a particular section of your canvas drawing.
12. This allows you to apply temporary changes without affecting the entire canvas, making it easier to manage complex drawings and maintain the desired state.

5. Canvas - Translation

- In HTML5 Canvas, translation refers to the process of moving the canvas origin to a different position.
- It allows you to shift the starting point (0, 0) of the coordinate system on the canvas.
- The translate() method is used to perform translation.
- Here's an example that demonstrates how to use translation in canvas:

Example

```
// Get the canvas element
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
```

```
// Perform translation
ctx.translate(100, 50);

// Draw a rectangle at the translated position
ctx.fillStyle = "red";
ctx.fillRect(0, 0, 100, 50);
```

In the example above:

1. We obtain a reference to the canvas element and its 2D rendering context.
2. We use the translate() method to move the origin point of the canvas by specifying the horizontal and vertical distances to translate.
3. In this case, we translate the canvas by 100 units horizontally and 50 units vertically.
4. After the translation, any subsequent drawing operations will be based on the translated coordinate system.
5. We draw a rectangle using fillRect() at the translated position (0, 0) on the canvas. Since the canvas has been translated, the rectangle will be drawn at the translated position (100, 50) on the original coordinate system.
6. By applying translation, you can effectively change the reference point of the canvas, allowing you to position and draw objects at different locations on the canvas.

6. Canvas -Rotation

- In HTML5 Canvas, rotation allows you to rotate the canvas or individual objects around a specific point.
- The rotate() method is used to perform rotation.
- Here's an example that demonstrates how to use rotation in canvas:

Example

```
// Get the canvas element
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");

// Set the rotation angle in radians
var angle = Math.PI / 4;

// Translate to the center of the canvas
ctx.translate(canvas.width / 2, canvas.height / 2);

// Rotate the canvas
ctx.rotate(angle);
// Draw a rectangle after rotation
ctx.fillStyle = "red";
ctx.fillRect(-50, -50, 100, 100);
```

In the example above:

1. We obtain a reference to the canvas element and its 2D rendering context.
2. We define the rotation angle in radians. In this case, we set it to π/4, which corresponds to a 45-degree rotation.
3. We use the translate() method to move the origin point of the canvas to the center of the canvas.
4. This ensures that the rotation occurs around the center.
5. We use the rotate() method to rotate the canvas by the specified angle.
6. After the rotation, any subsequent drawing operations will be based on the rotated coordinate system.
7. We draw a rectangle using fillRect() at the position (-50, -50) relative to the rotated coordinate system.
8. This rectangle will be drawn rotated by 45 degrees around the center of the canvas.
9. By applying rotation, you can change the orientation of the canvas or individual objects, allowing you to create dynamic and visually interesting graphics on the canvas.

7. Canvas - Scaling

- In HTML5 Canvas, scaling allows you to resize and scale the canvas or individual objects.
- The scale() method is used to perform scaling.
- Here's an example that demonstrates how to use scaling in canvas:

Example

```
// Get the canvas element
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");

// Scale the canvas
ctx.scale(2, 2);

// Draw a rectangle after scaling
ctx.fillStyle = "red";
ctx.fillRect(50, 50, 100, 50);
```

In the example above:

1. We obtain a reference to the canvas element and its 2D rendering context.
2. We use the scale() method to resize the canvas by specifying the horizontal and vertical scaling factors.
3. In this case, we scale the canvas by a factor of 2 in both directions, effectively doubling its size.
4. After the scaling, any subsequent drawing operations will be based on the scaled coordinate system.

5. We draw a rectangle using fillRect() at the position (50, 50) on the scaled canvas.
6. The dimensions of the rectangle are also affected by the scaling, resulting in a rectangle that is twice as large as it would be without scaling.
7. Scaling can also be applied to individual objects on the canvas, allowing you to resize and transform specific elements independently.
8. By adjusting the scaling factors, you can increase or decrease the size of the canvas or objects within it, creating interesting visual effects and transformations.

8. Canvas - Transforms

- In HTML5 Canvas, transforms allow you to perform complex transformations on the canvas, including translation, rotation, scaling, and arbitrary transformations.
- The transform() and setTransform() methods are used to apply transformations.
- Here's an example that demonstrates how to use transforms in canvas:

Example

```
// Get the canvas element
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");

// Perform multiple transformations
ctx.translate(100, 100);
ctx.rotate(Math.PI / 4);
ctx.scale(2, 2);

// Draw a rectangle after transformations
ctx.fillStyle = "red";
ctx.fillRect(0, 0, 50, 50);
```

In the example above:

1. We obtain a reference to the canvas element and its 2D rendering context.
2. We use the translate() method to move the origin point of the canvas by (100, 100).
3. This will shift subsequent drawing operations.
4. We use the rotate() method to rotate the canvas by an angle of π/4 (45 degrees) around the origin point.
5. We use the scale() method to scale the canvas by a factor of 2 in both directions.
6. After the transformations, any subsequent drawing operations will be based on the transformed coordinate system.
7. We draw a rectangle using fillRect() at the position (0, 0) on the transformed canvas.
8. The rectangle will be affected by the translations, rotations, and scaling applied.
9. By combining translations, rotations, and scaling, you can create complex transformations to position and manipulate objects on the canvas.

10. Additionally, the setTransform() method allows you to set the transformation matrix directly, providing even more control over transformations.

9. HTML5 Canvas -Composition

- In HTML5 Canvas, composition refers to the way newly drawn shapes are blended with the existing content on the canvas.
- The globalCompositeOperation property is used to set the blending mode for rendering elements.
- Here's an example that demonstrates how to use composition in Canvas:

Example

```
// Get the canvas element
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");

// Set the composition mode
ctx.globalCompositeOperation = "destination-over";

// Draw a blue rectangle
ctx.fillStyle = "blue";
ctx.fillRect(50, 50, 200, 100);

// Draw a red circle
ctx.fillStyle = "red";
ctx.beginPath();
ctx.arc(200, 150, 50, 0, 2 * Math.PI);
ctx.fill();
```

In the example above:

1. We obtain a reference to the canvas element and its 2D rendering context.
2. We set the globalCompositeOperation property to "destination-over".
3. This composition mode causes new shapes to be drawn behind the existing content on the canvas.
4. We draw a blue rectangle using fillRect().
5. This rectangle will be drawn as the background.
6. We draw a red circle using fill(). Since the composition mode is set to "destination-over", the circle will be drawn behind the existing blue rectangle.
7. The globalCompositeOperation property allows you to set various blending modes such as "source-over", "source-in", "source-out", "destination-over", "destination-in", "destination-out", and more.
8. Each blending mode produces a different result when drawing new shapes on the canvas.

9. By changing the composition mode, you can create complex compositions, layering effects, and compositing operations to achieve the desired visual effects on the canvas.
10. Experiment with different composition modes and drawing operations to create interesting and unique compositions.

10. Canvas – Animations

- In HTML5 Canvas, you can create animations by redrawing the canvas multiple times per second.
- Animations in Canvas are typically achieved by utilizing techniques like request AnimationFrame or setInterval to repeatedly update and render the canvas content.
- Here's a basic example of creating an animation in Canvas:

Example

```
// Get the canvas element
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");

// Set initial position
var x = 0;

// Update and render the animation
function animate() {
  // Clear the canvas
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  // Draw the animated object
  ctx.fillStyle = "red";
  ctx.fillRect(x, 50, 50, 50);

  // Update the position
  x += 1;

  // Request the next animation frame
  requestAnimationFrame(animate);
}

// Start the animation
animate();
```

In the example above:

1. We obtain a reference to the canvas element and its 2D rendering context.
2. We set an initial position for the animated object (a red rectangle in this case).

3. The animate() function is defined to update and render the animation.
4. Within the animate() function, we first clear the canvas using clearRect() to remove the previous frame.
5. We draw the animated object (a red rectangle) at the current position.
6. We update the position of the object to create the animation effect. In this case, we increment the x value by 1 in each frame to move the rectangle horizontally.
7. Finally, we request the next animation frame using requestAnimationFrame(animate). This recursively calls the animate() function to create a continuous animation loop.