**Πηγή :** [**https://www.youtube.com/watch?v=Xh-wIMqohD0&t=16s**](https://www.youtube.com/watch?v=Xh-wIMqohD0&t=16s)

**<!DOCTYPE html>**

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

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

<title>Document</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<div class="slider">

<div class="item">

<h1>Scream IV</h1>

<img src="scream-6-poster-small.jpg" alt="Movie" width="200" height="250">

</div>

<div class="item">

<h1>65</h1>

<img src="image.jpg" alt="Movie" width="200" height="250">

</div>

<div class="item">

<h1>Guardians of the Galaxy 3</h1>

<img src="disney-movies-2023-guardians-of-the-galaxy-1673546742.jpeg" alt="Movie" width="200" height="230">

</div>

<div class="item">

<h1>John Wick 4</h1>

<img src="ImageGen.jpg" alt="Movie" width="200" height="250">

</div>

<div class="item">

<h1>Dungeons and Dragons</h1>

<img src="new-family-movies-dungeons-and-dragons.png" alt="Movie" width="200" height="230">

</div>

<div class="item">

<h1>Winnie the Pooh</h1>

<img src="2023-horror-movies-winnie-the-pooh-1672365534.jpg" alt="Movie" width="200" height="230">

</div>

<div class="item">

<h1>M3GAN</h1>

<img src="best-new-horror-movies-m3gan-1670003739.jpg" alt="Movie" width="200" height="250">

</div>

<button id="next">></button>

<button id="prev"><</button>

</div>

<script src="app.js"></script>

</body>

</html>

**CSS**

body{

background-image:

linear-gradient(

to top right, #8B5CF6, #EC4899

);

min-height: 100vh;

margin: 0;

display: flex;

justify-content: center;

align-items: center;

font-family: monospace;[[1]](#footnote-1)

}[[2]](#footnote-2)

.slider{

position: relative;[[3]](#footnote-3)

width: 100%;

height: 370px;

overflow: hidden;

}

.item{

position: absolute;

width: 200px;

height: 320px;

text-align: justify;

background-color: #fff;

border-radius: 10px;

padding: 20px;

transition: 0.5s;

left: calc(50% - 110px);

top: 0;

}

#next, #prev{

position: absolute;[[4]](#footnote-4)

top: 40%;

color: #fff;

background-color: transparent;[[5]](#footnote-5)

border: none;

font-size: xxx-large;[[6]](#footnote-6)

font-family: monospace;

font-weight: bold;

left: 400px;

}

#next{

left: unset;

right: 400px;

}

**JavaScript**

let items = document.querySelectorAll('.slider .item');[[7]](#footnote-7)

let next = document.getElementById('next');[[8]](#footnote-8)

let prev = document.getElementById('prev');[[9]](#footnote-9)

let active = 3;

function loadShow(){

let stt = 0;

items[active].style.transform = `none`;

items[active].style.zIndex = 1;

items[active].style.filter = 'none';

items[active].style.opacity = 1;

for(var[[10]](#footnote-10) i = active + 1; i < items.length; i++)[[11]](#footnote-11){

stt++;

items[i].style.transform = `translateX(${120\*stt}px) scale(${1 - 0.2\*stt}) perspective(16px) rotateY(-1deg)`;[[12]](#footnote-12)

items[i].style.zIndex = -stt;

items[i].style.filter = 'blur(5px)';

items[i].style.opacity = stt > 2 ? 0 : 0.6;

}

stt = 0;

for(var i = active - 1; i >= 0; i--){

stt++;

items[i].style.transform = `translateX(${-120\*stt}px) scale(${1 - 0.2\*stt}) perspective(16px) rotateY(1deg)`;

items[i].style.zIndex = -stt;

items[i].style.filter = 'blur(5px)';

items[i].style.opacity = stt > 2 ? 0 : 0.6;

}

}[[13]](#footnote-13)

This code is animating a carousel of items, where the currently active item is in the center and the previous and next items are being translated and rotated to give the impression of a 3D perspective.

The code first selects the items in the carousel and uses a for loop to iterate through them. For each item, it calculates the position and style attributes needed to create the 3D effect using the translateX(), scale(), perspective(), and rotateY() CSS transform functions. It also adjusts the zIndex, filter, and opacity properties to ensure that the items are positioned and displayed correctly.

The first loop moves the items to the right of the currently active item, while the second loop moves the items to the left of the currently active item. The variable stt is used to keep track of the order of the items as they move away from the center. The blur() function is used to create a blur effect on the items as they move away from the center to give the impression of depth. The opacity property is set to gradually fade the items as they move further away from the center.

Overall, this code creates a visually appealing carousel animation effect by applying 3D transformations to the items in the carousel.

The first for loop is iterating through the items in the carousel starting from the item after the currently active item (as indicated by the active variable). For each item, it increments the stt variable and uses it to calculate the translateX(), scale(), perspective(), and rotateY() CSS transform functions. It also sets the zIndex property to -stt, which means that the items further away from the center of the carousel will have a lower z-index value and appear behind the items closer to the center. The filter property is set to apply a blur() effect to the item, which will make it appear further away in the 3D perspective. Finally, the opacity property is set to gradually fade the items as they move further away from the center of the carousel.

After the first loop finishes, the stt variable is reset to zero, and the second loop iterates through the items in the carousel before the currently active item. It performs similar transformations to the first loop, but the translateX() and rotateY() functions are adjusted to move the items in the opposite direction. The zIndex, filter, and opacity properties are also set in the same way as the first loop.

Overall, this code creates an animation effect that makes the items in the carousel appear to move in a 3D perspective, with the currently active item in the center and the previous and next items moving away from it.

loadShow();

next.onclick = function(){

active = active + 1 < items.length ? active + 1 : active;

loadShow();

}[[14]](#footnote-14)

prev.onclick = function(){

active = active - 1 >= 0 ? active - 1 : active;

loadShow();

}[[15]](#footnote-15)

1. The CSS rule "font-family: monospace;" sets the font family of an HTML element to be a typeface that is commonly referred to as monospace or fixed-width font.

   Monospace fonts are characterized by having a consistent width for each character, regardless of the actual size of the character. This means that each character takes up the same amount of space, which makes them useful for programming or displaying data in columns or tables where alignment is important.

   Some common examples of monospace fonts include Courier, Consolas, and Monaco.

   Using "monospace" as the font family can be useful in situations where you want to display code, terminal output, or other types of fixed-width text content. It can also be used as a stylistic choice to create a retro or vintage feel to the text. [↑](#footnote-ref-1)
2. The "background-image" property sets a linear gradient background for the body element. The gradient is created using the "linear-gradient" function and starts at the bottom left corner and goes to the top right corner. The first color of the gradient is #8B5CF6 and the second color is #EC4899.

   The "min-height" property sets the minimum height of the body element to 100% of the viewport height. This ensures that the content of the body element covers at least the entire height of the viewport.

   The "margin" property sets the margin of the body element to 0, which removes any default margin that may be applied to the body element.

   The "display" property sets the display style of the body element to "flex", which makes it a flex container.

   The "justify-content" property aligns the child elements within the flex container along the horizontal axis, in this case, it centers them horizontally.

   The "align-items" property aligns the child elements within the flex container along the vertical axis, in this case, it centers them vertically.

   The "font-family" property sets the font family for the text content within the body element to "monospace". [↑](#footnote-ref-2)
3. The CSS rule "position: relative" sets the positioning of an HTML element to be relative to its normal position within the document flow.

   When an element has "position: relative" set, it is positioned relative to its normal position within the document flow, meaning that it will be displayed in its normal position until you apply a CSS rule that changes its position.

   This rule is often used in conjunction with other positioning properties, such as "top", "bottom", "left", and "right", which can be used to move the element relative to its normal position.

   One of the key advantages of using "position: relative" is that it allows you to position an element relative to its parent container, which can be useful for creating layouts that are responsive to changes in screen size or device orientation.

   Overall, "position: relative" is a flexible and powerful CSS rule that can be used in a variety of ways to create sophisticated page layouts and responsive designs. [↑](#footnote-ref-3)
4. An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

   However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

   Note: Absolute positioned elements are removed from the normal flow, and can overlap elements. [↑](#footnote-ref-4)
5. The CSS rule "background-color: transparent" sets the background color of an HTML element to be transparent, meaning that it is completely see-through and any content behind the element will be visible.

   This can be useful when you want to have a background image or a gradient displayed behind an element but still be able to see the content within the element. By setting the background color to transparent, the background image or gradient will be visible through the element, creating a visually appealing effect.

   It is also commonly used when working with overlays, modals, or other types of pop-up elements that need to be displayed over other content on the page while still allowing the content underneath to be visible. [↑](#footnote-ref-5)
6. The CSS rule "font-size: xxx-large;" sets the font size of an HTML element to be very large.

   The "xxx-large" value is a relative size keyword that represents a very large font size, which is typically larger than the "xx-large" size and smaller than the "larger" size.

   The exact size of "xxx-large" can vary depending on the user's browser and default font size settings, but it is generally larger than the default font size.

   Using this rule can be useful when you want to make a particular heading or text element stand out on a page, especially if you want it to be significantly larger than other text on the page. However, it is important to use it judiciously, as using very large font sizes can make text difficult to read and may negatively affect the overall usability and accessibility of the page. [↑](#footnote-ref-6)
7. The JavaScript code "let items = document.querySelectorAll('.slider .item');" selects all the HTML elements with the class "item" that are descendants of an element with the class "slider" and returns a collection of these elements.

   The document.querySelectorAll() method is used to select elements from the HTML document using CSS selectors. In this case, the .slider .item selector is used to target all elements with the class "item" that are descendants of an element with the class "slider".

   The let keyword is used to declare a new variable called "items" that is assigned the collection of HTML elements that match the selector. This variable can then be used to manipulate or traverse the selected elements using JavaScript methods and properties.

   Overall, this code is useful for selecting and manipulating a group of related elements on a webpage, such as a group of images or other content in a slideshow or carousel. [↑](#footnote-ref-7)
8. The JavaScript code "let next = document.getElementById('next');" selects the HTML element with the ID "next" and assigns it to a new variable called "next".

   The document.getElementById() method is used to select a single HTML element from the document using its unique ID. In this case, the "next" variable is assigned to the HTML element that has the ID "next".

   Once assigned to the variable, the "next" element can be manipulated using JavaScript methods and properties, such as adding event listeners or changing its style attributes.

   This code can be useful for selecting and manipulating a specific HTML element on a webpage, such as a button or link, and can be used to create dynamic and interactive user interfaces. [↑](#footnote-ref-8)
9. The JavaScript code "let prev = document.getElementById('prev');" selects the HTML element with the ID "prev" and assigns it to a new variable called "prev".

   The document.getElementById() method is used to select a single HTML element from the document using its unique ID. In this case, the "prev" variable is assigned to the HTML element that has the ID "prev".

   Once assigned to the variable, the "prev" element can be manipulated using JavaScript methods and properties, such as adding event listeners or changing its style attributes.

   This code can be useful for selecting and manipulating a specific HTML element on a webpage, such as a button or link, and can be used to create dynamic and interactive user interfaces. In this case, "prev" could be used to create a button or link that allows users to move backwards in a slideshow or carousel, for example. [↑](#footnote-ref-9)
10. Always declare JavaScript variables with var,let, or const.

    The var keyword is used in all JavaScript code from 1995 to 2015.

    The let and const keywords were added to JavaScript in 2015.

    If you want your code to run in older browsers, you must use var. [↑](#footnote-ref-10)
11. for - loops through a block of code a number of times

    The for statement creates a loop with 3 optional expressions:

    for (expression 1; expression 2; expression 3) {

    // code block to be executed

    }

    Expression 1 is executed (one time) before the execution of the code block.

    Expression 2 defines the condition for executing the code block.

    Expression 3 is executed (every time) after the code block has been executed. [↑](#footnote-ref-11)
12. The translateX() function is being used to move the element horizontally along the x-axis by 120\*stt pixels, where stt is a variable that keeps track of the position of the element in the carousel.

    The scale() function is being used to decrease the size of the element as it moves further away from the center of the carousel. The 1 - 0.2\*stt expression is used to calculate the scale factor, where stt is a variable that increments for each item further away from the center. This expression will decrease the scale factor by 0.2 for each step away from the center.

    The perspective() function is used to define the depth of the 3D perspective effect. In this case, a perspective of 16px is used.

    Finally, the rotateY() function is used to rotate the element around the y-axis by -1deg. This will create a slight angle in the 3D perspective effect.

    Overall, this line of code is setting a complex 3D transformation on the element to create a visually appealing carousel animation effect. [↑](#footnote-ref-12)
13. This JavaScript code defines a function called loadShow() which performs various transformations on a collection of HTML elements selected by the items variable. The function also uses the active variable to determine which element in the collection is currently active.

    The function begins by setting the transform, z-index, filter, and opacity properties of the active element to specific values, effectively resetting it to its default state.

    Next, the function iterates over the elements in the items collection that come after the active element, applying transform, z-index, filter, and opacity properties to each one that create a sliding and scaling effect, as well as a perspective and rotation effect. The opacity of each element decreases as it moves further from the active element.

    Finally, the function iterates over the elements in the items collection that come before the active element, applying similar transform, z-index, filter, and opacity properties to each one, but with the opposite direction and rotation. The opacity of each element decreases as it moves further from the active element.

    The overall effect of the loadShow() function is to create a visually appealing slideshow or carousel effect, where the active element is centered and larger, with the surrounding elements scaled and rotated to give the appearance of depth and motion. [↑](#footnote-ref-13)
14. This code is defining a click event handler for a "next" button, which is represented by the next variable.

    When the button is clicked, the active variable is incremented by 1 if it is less than the number of items in the carousel (items.length). However, if active is already at the last item in the carousel, it is not incremented any further and remains at its current value.

    After updating the active variable, the loadShow() function is called. This function is most likely responsible for updating the display of the carousel based on the new active value, which will shift the carousel one item to the right.

    In summary, this code allows the user to move the carousel one item to the right when the "next" button is clicked, as long as there are more items to display. If the last item in the carousel is already active, the button does not do anything. [↑](#footnote-ref-14)
15. This code is defining a click event handler for a "previous" button, which is represented by the prev variable.

    When the button is clicked, the active variable is decremented by 1 if it is greater than or equal to zero. However, if active is already at the first item in the carousel, it is not decremented any further and remains at its current value.

    After updating the active variable, the loadShow() function is called. This function is most likely responsible for updating the display of the carousel based on the new active value, which will shift the carousel one item to the left.

    In summary, this code allows the user to move the carousel one item to the left when the "previous" button is clicked, as long as the current active item is not already the first item. If the first item in the carousel is already active, the button does not do anything. [↑](#footnote-ref-15)