LEARN CSS TRANSFORMS BY BUILDING A PENGUIN

Introduction:

You can transform HTML elements to create appealing designs that draw your reader's eye. You can use transforms to rotate elements, scale them, and more.

In this course, you'll build a penguin. You'll use CSS transforms to position and resize the parts of your penguin, create a background, and animate your work.

Step 1:

You will be building a happy Flappy Penguin, and further exploring CSS transforms and animations in the process.

Begin with your basic HTML boilerplate. Include the `DOCTYPE` declaration, `html` element with a language set to English, the appropriate `meta` tags, a `head`, `body`, and `title` element. Also, link your stylesheet to the page.

Step 2:

Target the `body` element to set the `background` to a linear gradient angled 45 degrees clockwise, starting at `rgb(118, 201, 255)` and ending at `rgb(247, 255, 222)`.

Step 3:

Normalise your page's sizing, by removing the `body` element's `margin` and `padding`.

Step 4:

Normalise your page, by setting the `width` to `100%`, and `height` to `100vh`.

Step 5:

Remove both the horizontal and vertical scrollbars, using only one property.

Step 6:

Within the `body`, add a `div` with a `class` of `ground`.

Step 7:

Target the `.ground` element, and set its `width` to take up the full width of the viewport. Then, set the `height` to `400px`.

Step 8:

Give the `.ground` element a `background` with a linear gradient angled 90 degrees clockwise, starting at `rgb(88, 175, 236)` and ending at `rgb(182, 255, 255)`.

Step 9:

As the `.ground` element will be third in the stacking context of the page layout, set its `z-index` to `3`, and `position` to `absolute`.

Step 10:

Above the `.ground` element, add a `div` with a `class` of `penguin`. This `div` will contain Flappy Penguin.

Step 11:

Target the `.penguin` element, and set its `width` and `height` to `300px`.

Step 12:

Use the `margin` property to horizontally center the `.penguin` element, and set the `margin-top` to `75px`.

Step 13:

To create some scenery in the background, you will add two mountains.

Above the `.penguin` element, add a `div` with a `class` of `left-mountain`.

Step 14:

Target the `.left-mountain` element, and set its `width` and `height` to `300px`. Then, set the `background` to a linear gradient starting at `rgb(203, 241, 228)` and ending at `rgb(80, 183, 255)`.

Step 15:

To prevent the mountain from pushing the `.ground` element, adjust its `position` to prevent it from taking up space in the page layout.

Step 16:

To make the mountain look more like a mountain, you can use the `skew` transform function, which takes two arguments. The first being an angle to shear the x-axis by, and the second being an angle to shear the y-axis by.

Use the `transform` property to skew the mountain by `0deg` in the x-axis and `44deg` in the y-axis.

Step 17:

Set the stack level of the mountain element such that it remains directly behind the `.ground` element.

Step 18:

To overlap the mountain and `.ground` elements better, give the mountain a `margin-top` of `100px`, and the `.ground` element a `margin-top` of `-58px`.

Step 19:

To give the effect of a mountain range, add another mountain, by creating a new `div` immediately after `.left-mountain`, and give the new `div` the `class` of `back-mountain`.

Step 20:

Target the `.back-mountain` element, and set its `width` and `height` to `300px`. Then, set the `background` to a linear gradient starting at `rgb(203, 241, 228)` and ending at `rgb(47, 170, 255)`.

Step 21:

Set the `position` property of the `.back-mountain` to prevent it from taking up space in the page layout.

Step 22:

Change the stack level of the `.back-mountain` element such that it is directly behind the `.left-mountain` element.

Step 23:

Rotate the `.back-mountain` element by `45deg` clockwise. Then, give it a `left` property of `110px`, and a `top` property of `225px`.

Step 24:

To finish the background, add a sun, by creating a new `div` element immediately after the `.back-mountain` element, and give it the class of `sun`.

Step 25:

Give the `.sun` element a `width` and `height` of `200px`, and a `background-color` of `yellow`.

Step 26:

Set the position property of the sun to prevent it from taking up space in the page layout, and set the border-radius such that the sun's shape is a circle.

Step 27:

Position the sun in the top right corner of the screen such that 75px of its top and right edges are off screen.

Step 28:

Your penguin will consist of two main sections: the head, and the body.

Within .penguin, add two new div elements. The first with a class of penguin-head, and the second with a class of penguin-body.

Step 29:

Change the stack level of the .penguin element such that it appears in front of the .ground element, and give it a position of relative.

Step 30:

Target the .penguin-head element, and give it a width half of its parent's, and a height of 45%. Then, set the background to a linear gradient at 45deg starting at gray, and ending at rgb(239, 240, 228).

Step 31:

Most penguins do not have a square head.

Give the penguin a slightly oval head by setting the radius of the top corners to 70% and the radius of the bottom corners to 65%.

Step 32:

Target the .penguin-body element, and give it a width of 53%, and a height of 45%. Then, set the background to a linear gradient at 45deg, rgb(134, 133, 133) from 0%, rgb(234, 231, 231) from 25%, and white from 67%.

Step 33:

Another interesting fact about penguins is that they do not have square bodies.

Use the border-radius property with a value of 80% 80% 100% 100%, to give the penguin a slightly rounded body.

Step 34:

Target all descendent elements of the .penguin element, and give them a position of absolute.

Step 35:

Position the .penguin-head element 10% from the top, and 25% from the left of its parent.

Step 36:

Position the .penguin-body element 40% from the top, and 23.5% from the left of its parent.

Step 37:

Change the stack level of the `.penguin-head` element such that it appears in front of the `.penguin-body` element.

Step 38:

To give the penguin body a crest, create a pseudo-element that is the first child of the `.penguin-body` element. Set the `content` property of the pseudo-element to an empty string.

Step 39:

Position the pseudo-element relative to its closest positioned ancestor.

Step 40:

Give the pseudo-element a `width` half that of its parent, a `height` of `45%`, and a `background-color` of `gray`.

Step 41:

Position the pseudo-element `10%` from the top and `25%` from the left of its parent.

Step 42:

Round off the crest, by giving the pseudo-element bottom corners a radius of `100%`, leaving the top corners at `0%`.

Step 43:

Increase the pseudo-element's transparency by 30%.

Step 44:

Start the penguin's face, by adding two div elements within .penguin-head, and giving them both a class of face.

Step 45:

Give the .face elements a width of 60%, a height of 70%, and a background-color of white.

Step 46:

Make the top corners of the .face elements have a radius of 70%, and the bottom corners have a radius of 60%.

Step 47:

Position the .face elements so that they are 15% from the top.

Step 48:

Currently, the two .face elements are on top of each other.

Fix this, by adding a class of left to the first .face element, and a class of right to the second .face element.

Step 49:

Target the `.face` element with the `left` class, and position it `5%` left of its parent.

Step 50:

Target the `.face` element with the `right` class, and position it `5%` right of its parent.

Step 51:

Below the `.face.right` element, add a `div` element with a `class` of `chin`.

Step 52:

Target the `.chin` element, and give it a `width` of `90%`, `height` of `70%`, and `background-color` of `white`.

Step 53:

Position the `.chin` element such that it is `25%` from the top, and `5%` from the left of its parent. Then, give the top corners a radius of `70%`, and the bottom corners a radius of `100%`.

Step 54:

So far, the `.face` and `.chin` elements have the same `background-color`.

Create a custom CSS property called `--penguin-face`, and set it to `white`.

Step 55:

Where relevant, replace property values with your `--penguin-face` variable.

Step 56:

Below the `.chin` element, add two `div` elements each with a `class` of `eye`. Also, give the first `.eye` element a `class` of `left`, and the second `.eye` element a `class` of `right`.

Step 57:

Target the `.eye` elements, and give them a `width` of `15%`, `height` of `17%`, and `background-color` of `black`.

Step 58:

Position the `.eye` elements `45%` from the top of their parent, and give all corners a radius of `50%`.

Step 59:

Target the `.eye` element with the `left` class, and position it `25%` from the left of its parent. Then, target the `.eye` element with the `right` class, and position it `25%` from the right of its parent.

Step 60:

Within each `.eye` element, add a `div` with a `class` of `eye-lid`.

Step 61:

Target the `.eye-lid` elements, and give them a `width` of `150%`, `height` of `100%`, and `background-color` of `--penguin-face`.

Step 62:

Position the `.eye-lid` elements `25%` from the top, and `-23%` from the left of their parents. Then, give all corners a radius of `50%`.

Step 63:

Below the `.eye.right` element, add two `div` elements each with a `class` of `blush`. Also, give the first `.blush` element a `class` of `left`, and the second `.blush` element a `class` of `right`.

Step 64:

Target the `.blush` elements, and give them a `width` of `15%`, `height` of `10%`, and `background-color` of `pink`.

Step 65:

Position the `.blush` elements `65%` from the top of their parent, and give all corners a radius of `50%`.

Step 66:

Target the .blush element with a class of left, and position it 15% left of its parent. Then, target the .blush element with a class of right, and position it 15% right of its parent.

Step 67:

Below the .blush.right element, add two div elements each with a class of beak. Also, give the first .beak element a class of top, and the second .beak element a class of bottom.

Step 68:

Target the .beak elements, and give them a height of 10%, background-color of orange, and give all corners a radius of 50%.

Step 69:

Target the .beak element with a class of top, give it a width of 20%, and position it 60% from the top, and 40% from the left of its parent.

Step 70:

Target the .beak element with a class of bottom, and give it a width 4% smaller than .beak.top, 5% further from the top, and 2% further from the left of its parent than .beak.top.

Step 71:

The penguin's body looks a bit plain. Spruce him up by adding a div element with a class of shirt, immediately before the .penguin-body element.

Step 72:

Within the `.shirt` element, add a `div` with the following emoji as content: 💜

Step 73:

Within `.shirt`, after the `div` element, add a `p` element with the following content: `I CSS.`

Step 74:

Target the `.shirt` element, and set its `font-size` to `25px`, `font-family` to `Helvetica` with a fallback of `sans-serif`, and `font-weight` to `bold`.

Step 75:

In some browsers, the *heart* emoji may look slightly different from the previous step. This is because some of the character's properties were overridden by the `font-weight` style of `bold`.

Fix this, by targeting the `div` with the heart emoji, and setting its `font-weight` to its original value.

Step 76:

Position the `div` with the heart emoji `22.5px` from the top, and `12px` from the left of its parent.

Step 77:

Position the .shirt element 165px from the top, and 127.5px from the left of its parent. Then, increase its stacking order such that it appears above the .penguin-body element.

Step 78:

For the shirt's final touch, set the color to #6a6969.

Step 79:

Fun fact: Penguins cannot stand without at least two feet.

Within the .penguin-body element, add two div elements each with a class of foot. Give the first .foot a class of left, and the second .foot a class of right.

Step 80:

Target the .foot elements, and give them a width of 15%, height of 30%, and background-color of orange.

Step 81:

Position the .foot elements 85% from the top of their parent, and give all corners a radius of 50%.

Step 82:

The penguin's beak and feet share the same color.

Create a new custom CSS variable named --penguin-picorna, and replace all relevant property values with it.

Step 83:

Target the .foot element with a class of left, and position it 25% left of its parent. Then, target the .foot element with a class of right, and position it 25% right of its parent.

Step 84:

To make the penguin's feet look more *penguiny*, rotate the left foot by 80deg, and the right by -80deg.

Step 85:

Change the stacking order of the .foot elements such that they appear beneath the .penguin-body element.

Step 86:

Fun fact: Penguins cannot fly without wings.

Within .penguin-body, before the .foot elements, add two div elements each with a class of arm. Give the first .arm a class of left, and the second .arm a class of right.

Step 87:

Target the .arm elements, and give them a width of 30%, a height of 60%, and a background of linear gradient at 90deg from clockwise, starting at gray, and ending at rgb(209, 210, 199).

Step 88:

Create a custom CSS variable named `--penguin-skin`, and set it to `gray`. Then, replace all relevant property values with it.

Step 89:

Target the `.arm` element with a `class` of `left`, and position it `35%` from the top, and `5%` from the left of its parent. Then, target the `.arm` element with a `class` of `right`, and position it `0%` from the top, and `-5%` from the right of its parent.

Step 90:

Within the `.arm.left` selector, alter the origin of the `transform` function to be the top left corner of its parent.

Step 91:

To keep the linear gradient on the correct side of the penguin's left arm, first rotate it by `130deg`, then invert the x-axis.

Step 92:

Rotate the right arm by `45deg` counterclockwise.

Step 93:

Fun fact: Most, if not all, flippers are not naturally rectangles.

Give the `.arm` elements' top-left, top-right, and bottom-right corners a radius of `30%`, and the bottom-left corner a radius of `120%`.

Step 94:

Change the `.arm` elements' stacking order such that they appear behind the `.penguin-body` element.

Step 95:

Now, you are going to use CSS animations to make the penguin wave.

Define a new `@keyframes` named `wave`.

Step 96:

Give `wave` four waypoints starting at `10%`, and incrementing by `10%`.

Step 97:

Within the first waypoint, rotate to `110deg`, and retain the scaling of the left arm.

Step 98:

Within the second waypoint, rotate to `130deg`, and retain the scaling of the left arm.

Step 99:

For the third and fourth waypoints, repeat the `transform` pattern once more.

Step 100:

Use the wave animation on the left arm. Have the animation last 3s, infinitely iterate, and have a linear timing function.

Step 101:

Target the .penguin element when it is active, and increase its size by 50% in both dimensions.

Step 102:

When you activate the .penguin element, it might look as though you can drag it around. This is not true.

Indicate this to users, by giving the active element a cursor property of not-allowed.

Step 103:

Change the .penguin element's transition behavior during transformation to have a duration of 1s, a timing function of ease-in-out, and a delay of 0ms.

Step 104:

Finally, calculate the height of the .ground element to be the height of the viewport minus the height of the .penguin element.

Congratulations! You have completed the Responsive Web Design certification.