COMP3322 MODERN TECHNOLOGIES ON WORLD WIDE WEB

Workshop - CSS Animation and JavaScript Event Handling

Overview

In this workshop, you will develop a simple Web page with a header that contains a line of text continuously moving from left to right across the screen. Below the header block, we have four blocks of information. Users can select which block of information to be displayed by clicking on the corresponding tag. Fig. 1 is the screen capture of this Web page. The amination of text is rendered by CSS and the selection of content is handled by JavaScript.

Fig. 1

Task 1: Examine the index.html Web document

Step1: We expect you use the course LAMP container set to work on the workshop. Download the "WK2-CSSEvt.zip" file from Moodle course site to the public_html folder of the c3322-WWW container and extract it to a folder name WK2. In this folder, you will find the index.html file, an images folder, the main.js, and style.css files. The index.html file

contains all the text contents of this Web page, while the main.js and style.css files are for the required JavaScript code and CSS rules.

Step2: Examine the index.html file. You should find that the body part contains a header block and a section block. In the section, it contains one div block for displaying the selection tags and four articles for displaying four different sets of contents. Each article contains an image and text.


```
</head>
<body>
  <header class='container header'>
    <h1>Workshop - CSS Animation and JavaScript Event Handling</h1>
    <div id='banner'>
      <p id='line'>WebAssembly brings the performance of native applications to the web.      
WebSockets are open connections sustained between the client and the origin HTTP server which enable data passing
between them.       The simplest use of workers is for performing a computationally expensive task
without interrupting the user interface.       A progressive web application is a type of application
software delivered through the web, built using common web technologies including HTML, CSS and JavaScript.</p>
    </div>
  </header>

  <section class='container'>
    <div id='selector'>
      <div class='menu blk1' id='blk1'>WebAssembly</div>
      <div class='menu blk2' id='blk2'>WebSocket</div>
      <div class='menu blk3' id='blk3'>Web Workers</div>
      <div class='menu blk4' id='blk4'>Progressive Web App</div>
    </div>
    <article class='blk1' id='block1'>
      <div class='infoblock'>
        <img class='img-icon' src='images/WASM-mid.png' alt='WebAssembly Icon'>
        <div>
        <p>WebAssembly is a new type of code that can be run in modern web browsers – it is a low-level assembly-like
language with a compact binary format that runs with near-native performance and provides languages such as C/C++, C#
and Rust with a compilation target so that they can run on the web.</p>
        <p>WebAssembly is designed to complement and run alongside JavaScript – using the WebAssembly JavaScript APIs,
you can load WebAssembly modules into a JavaScript app and share functionality between the two. This allows you to
take advantage of WebAssembly's performance and power and JavaScript's expressiveness and flexibility in the same
apps, even if you don't know how to write WebAssembly code.</p>
        </div>
      </div>
    </article>

    <article class='blk2' id='block2' style='display: none;'>
      <div class='infoblock'>
        <img class='img-icon' src='images/websocket-mid.png' alt='WebSocket Icon'>
        <div>
        <p>The idea of WebSockets was borne out of the limitations of HTTP-based technology. With HTTP, a client
requests a resource, and the server responds with the requested data. HTTP is a strictly unidirectional protocol – any
data sent from the server to the client must be first requested by the client. Long-polling has traditionally acted as
a workaround for this limitation. With long-polling, a client makes an HTTP request with a long timeout period, and
the server uses that long timeout to push data to the client. Long-polling works, but comes with a drawback –
resources on the server are tied up throughout the length of the long-poll, even when no data is available to send.</
p>
        <p>WebSockets, on the other hand, allow for sending message-based data, similar to UDP, but with the
reliability of TCP. WebSocket uses HTTP as the initial transport mechanism, but keeps the TCP connection alive after
the HTTP response is received so that it can be used for sending messages between client and server. WebSockets allow
us to build "real-time" applications without the use of long-polling.</p>
        </div>
      </div>
    </article>

    <article class='blk3' id='block3' style='display: none;'>
      <div class='infoblock'>
```

Try to understand the structure, purpose, and content of the individual block.

Task 2: Apply CSS styling

Open the style.css file with your code editor.

Step3: Add the basic styling for the header block – set the background color, the box shadow, and the text alignment.

```
.header {
  background-color: #f5f5f5;
  box-shadow: 0 4px 2px #c0c0c0;
  text-align: center;
}
```

Step4: Using the container class to set the styling of the header and section blocks. Set the font and make them appear in the middle of the page with 80% of the window width.

```
.container {
  font: 400 15px/1.8 "Lato", sans-serif;
  width: 80%;
  margin: 0 auto;
}
```

Next, we are going to create the text animation effect. The basic idea is that we have a paragraph of text, which scroll through the header block from the right edge to the left edge. To do so, we display the paragraph as a single line of text and adjust the margin-left property to simulate the effect of moving the text from right to left.

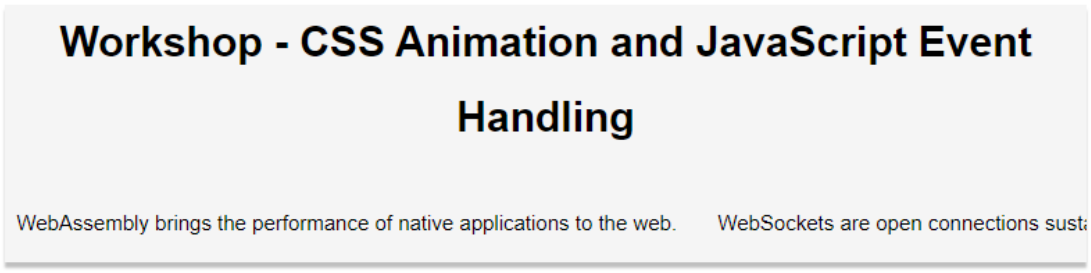
Step5: Display the paragraph #line as a single line.

```
#line {
  width: max-content;
}
```

Step6: Clip the #line content so that it will not be displayed outside of the header block.

```
#banner {
  padding: 0 0.5rem;
  overflow: hidden;
}
```

Access your page at http://localhost:9080/WK2/, the header block would look as follows:



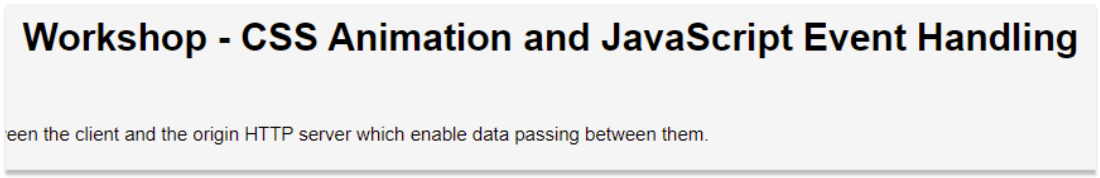
Step7: Add the text animation. CSS Animations is a module of CSS that lets us animate many CSS properties such as color, background-color, height, or width over time, using the @keyframes at-rule. The **CSS property** we going to adjust/animate is the **margin-left property** of the target line. However, one major issue is that we do not know the total length of the line. Without the information, we may not be able to control the final value of the margin-left property at the end of the animation and the total duration of the animation. Therefore, we need the help of JavaScript to create the best animation effect. Let's set up the basic CSS setting first.

```
:root {
  --ani-duration: 20s;
  --ani-start-margin: 100%;
  --ani-end-margin: -100%;
}

#line {
  animation-name: moveleft;
  animation-duration: var(--ani-duration);
  animation-timing-function: linear;
  animation-iteration-count: infinite;
}

@keyframes moveleft {
  from {
    margin-left: var(--ani-start-margin);
  }
  to {
    margin-left: var(--ani-end-margin);
  }
}
```

To use CSS animation, we must give a name to the animation and set the @keyframes at-rule to define what should happen at specific moments/steps during the animation. As we only know the exact setting of the animation during runtime, we use the CSS variables to define the initial values of the setting. We place the CSS variables in the root scope so that we can access the variables anywhere during the runtime.

In our case, our animation is named moveleft which is declared via the property animation-name. We define two moments/steps in the @keyframes rule. The "from" value defines the starting value of the margin-left property and the "to" value defines the ending value of the margin-left property in the animation. The animation starts from a position with margin-left sets to 100% of the parent container and ends the animation with the margin-left sets to negative 100%, which means the line is moved out of the parent container to the left. Please note that a part of the line may still be appeared within the parent container, especially when

the line is longer than the width of the parent container. For example,



We use the animation-duration property to set the total duration of the animation running in one loop to 20 seconds and use the animation-iteration-count property to set the repeat count to infinite. Lastly, use the animation-timing-function property to set how the animation progress in each loop.

You can get more information on CSS animation over here - https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations.

Step8: Set the selection menu. Use CSS flexbox layout to organize the menu tags horizontally.

```
section.container {
  margin-top: 2rem;
}

#selector {
  display: flex;
  justify-content: flex-start;
}
```

Define the appearance and background-color of each menu tag. The height and width of each tag are 2rem and 6rem respectively. Tags are separated by a margin gap of 1rem. The top left and right corners of the tags are set to round-corner.

```
.menu {
  text-align: center;
  height: 2rem;
  padding: 0.7rem 1rem 0.5rem 1rem;
  border-top-left-radius: 20px;
  border-top-right-radius: 20px;
  margin-right: 1rem;
  flex: 0 1 6rem;
  line-height: 1;
}

.blk1 {
  background-color: ghostwhite;
}
```

```
.blk2 {
  background-color: honeydew;
}
.blk3 {
  background-color: lemonchiffon;
}
.blk4 {
  background-color: seashell;
}
```

Now the selection menu would look as follows:

Step9: Set the appearance of the article block. First, set the article block with padding and text alignment.

```
article {
  padding: 30px 40px;
  text-align: justify;
}
```

Set the image icon to limit its display width and add some gaps to the left and bottom margins.

```
.img-icon {
  width: 220px;
  margin-left: 2rem;
  margin-bottom: 1rem;
}
```

Lastly, organize the text content and image icon with the flexbox layout. Move the image to the right by using flex-direction: row-reverse.

```
.infoblock {
  display: flex;
  flex-direction: row-reverse;
  align-items: flex-start;
}
```

After this step, the current look of the Web page will be like Fig 1.

Task 3: Using JavaScript to adjust CSS animation setting and implement content selection

Open the main.js file with your code editor.

Step10: Adjust CSS text animation.
Once the browser completely loads the Web page, we use JavaScript to detect (i) the width of the header block and (ii) the total length of the text line. Based on the relative ratio of the header width and line length, we determine the position of the margin-left property at the end of each animation loop and the total duration of the animation for each cycle. The basic idea is that we would like to end each animation cycle with the last word of the line gets to the middle of the header block and the rate of moving the text is around 100px per second.

```
/* adjust CSS animation */
var adjust = () => {
  let line = document.querySelector('#line');
  let wwidth = Math.ceil(window.innerWidth*0.8);
  let lwidth = line.scrollWidth;
  let end;
  let duration;
  if (lwidth < wwidth) {
    end = 0;
    duration = Math.ceil(wwidth/100);
  } else {
    end = (lwidth - Math.ceil(wwidth*0.5))*-1;
    duration = Math.ceil(lwidth/100);
  }
  let root = document.querySelector(':root');
  root.style.setProperty('--ani-end-margin', end+'px');
  root.style.setProperty('--ani-duration', duration+'s');
};

adjust();
```

Step11: Adjust CSS animation when the user resizes the browser window.

```
window.addEventListener('resize', (evt) => {
  adjust();
});
```

The browser calls the adjust() function once the user adjusts the browser's width.

Step12: Install an event handler for handling the user's selection.

By default, the browser only displays the first article and the rest are hidden by the “display: none” CSS setting. We would like to install only one click event handler for the selector div block and base on the target id to switch between which content block to be displayed.

```
/* select the block to show */
let selection = 'blk1';
let selected = 'block1';
let select = document.getElementById('selector');
select.addEventListener('click', (evt) => {
  if (evt.target.id == selection) {
    return;
  }
  switch (evt.target.id) {
    case 'blk1':
      document.getElementById(selected).style.display = 'none';
      selection = 'blk1';
      selected = 'block1';
      document.getElementById('block1').style.display = 'block';
      break;
    case 'blk2':
      document.getElementById(selected).style.display = 'none';
      selection = 'blk2';
      selected = 'block2';
      document.getElementById('block2').style.display = 'block';
      break;
    case 'blk3':
      document.getElementById(selected).style.display = 'none';
      selection = 'blk3';
      selected = 'block3';
      document.getElementById('block3').style.display = 'block';
      break;
    case 'blk4':
      document.getElementById(selected).style.display = 'none';
      selection = 'blk4';
      selected = 'block4';
      document.getElementById('block4').style.display = 'block';
      break;
  }
});
```

Now the user can use the selection menu to select which block of content to be displayed.

Test your page

Browse and test your Web page with different screen sizes at:
http://localhost:9080/WK2/