Slide Note

# Have you noticed how websites are made interactive in different ways?

Slide Note

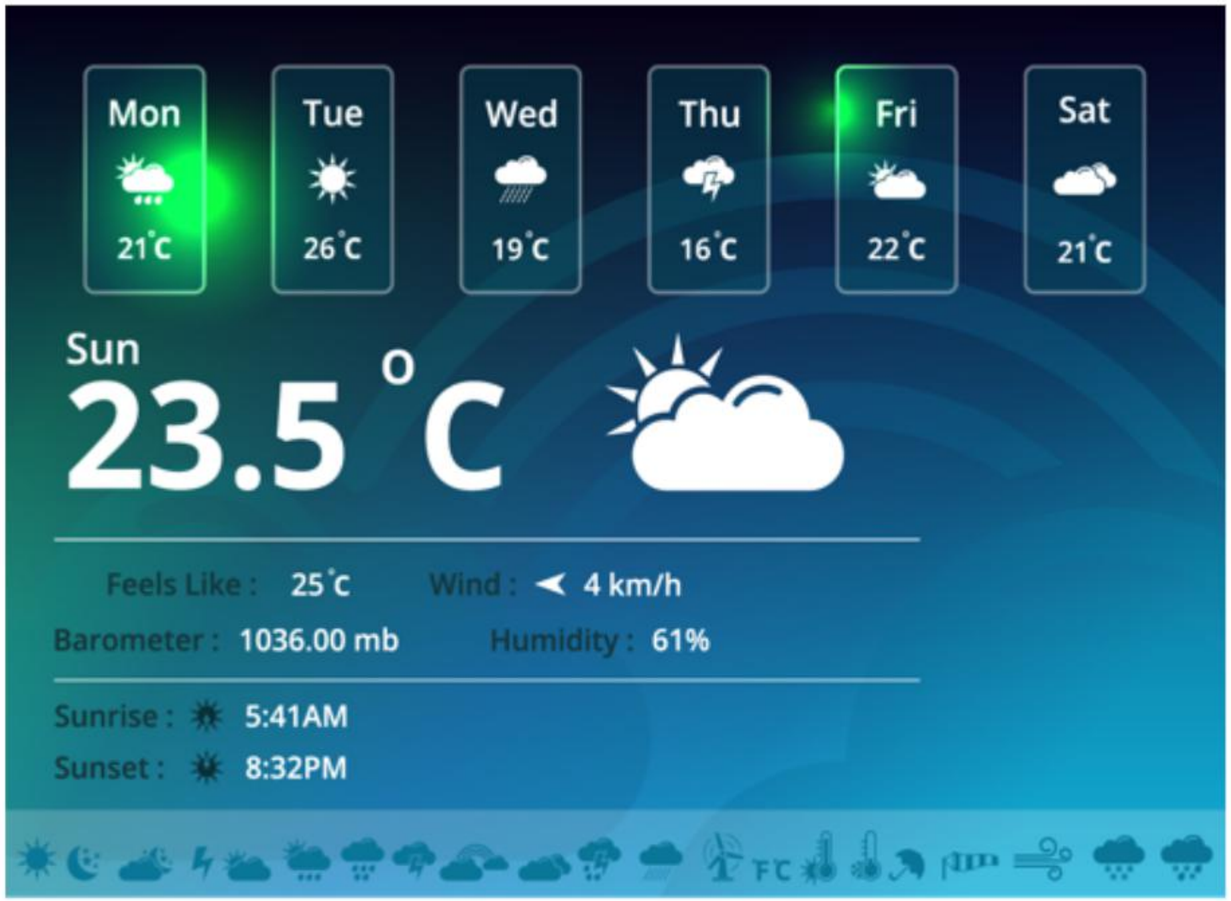
# Let's see how the background of a weather app changes with the time slider.

Menu

00:02:00:00:45

02/ 45

Slide Note


How does the background change when the date or time changes?

Slide Note

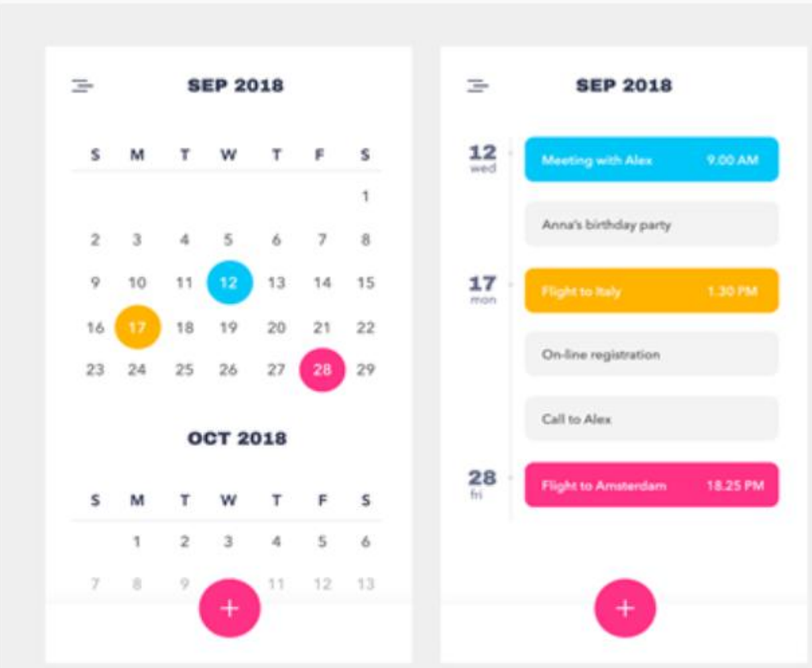
# Let's observe how the calendar slides from April to May.

Slide Note

How does the calendar slide from the month of April to May?

How does an Event Planner appear when a date is clicked?

# How Can We Add Events to the Calendar?


Menu

00:05:00:00:45 05/ 45

Slide Note

# Look at the image that shows how the page view changes with the change in the theme.

Slide Note

How does the view of the page change when we choose a different theme?

# How Do Websites Toggle Between Different Themes?

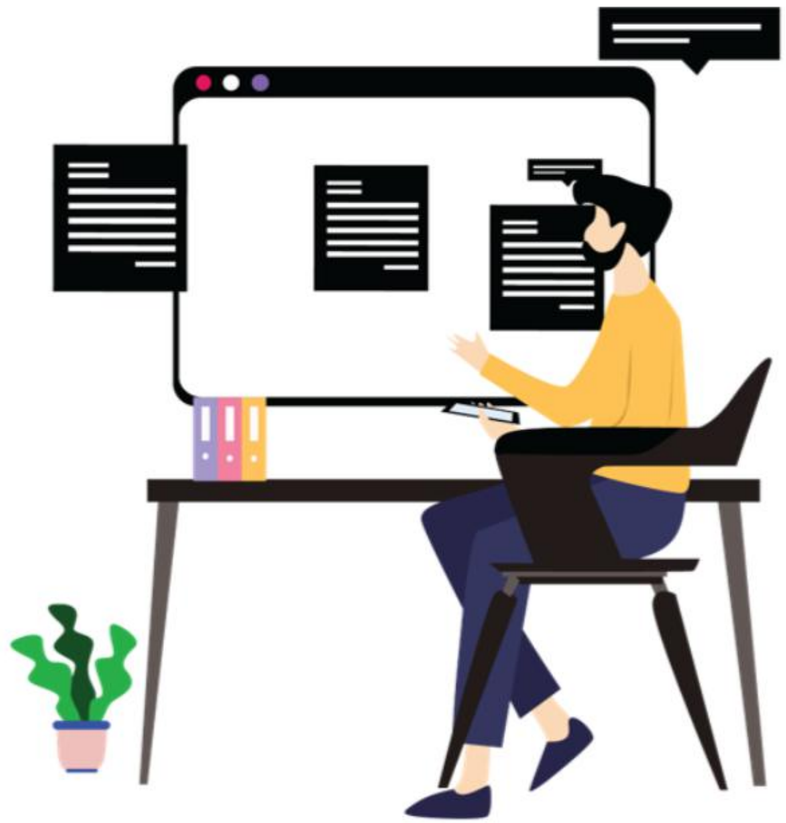
Slide Note

# Wonder how these changes are made?

# Do you want to learn how to make your web pages interactive?

Slide Note

# Develop Interactive Web Pages Using DOM and DOM Events



Slide Note

# Learning Objectives

- Explore Document Object Model (DOM) elements
- Manipulate DOM elements to add dynamic effects
- Access or change the style of the elements using DOM object properties
- Handle events by an action when a user interacts with a program
- Perform form validations using HTML5 built-in functionality

Slide Note

Event is an action that occurs as a result of the user or another source, such as a mouse click.

# What Is DOM (Document Object Model)?

- DOM is a standard for modelling web documents as objects that manipulates them using a standard programming interface.
- DOM defines:
  - The HTML elements as objects.
  - The properties and methods to access all HTML elements.
  - The events of all HTML elements.
- DOM tells you how to get, change, add or delete HTML elements dynamically which, makes the web page interactive.

**Note**: An event is an action that occurs because of the user or another source, such as a mouse click.

Slide Note

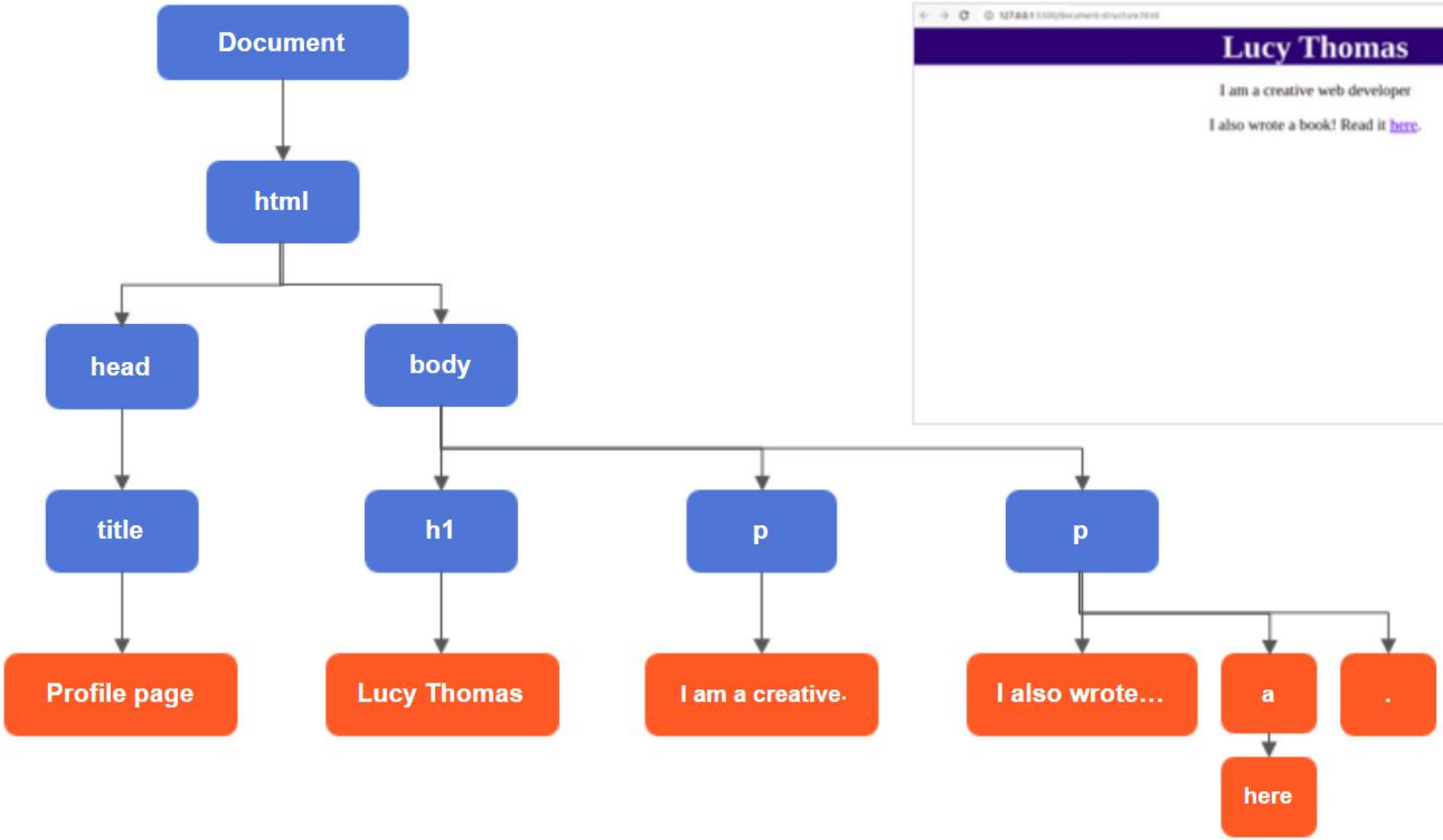
# Document Structure: Simple Web Page

```
<!doctype html>
<html>
  <head>
    <title>Profile Page</title>
  </head>
  <body>
    <header>
      <h1>Lucy Thomas</h1>
    </header>
    <section>
      <p>I am a creative web developer</p>
      <p>I also wrote a book! Read it
        <a href="basicweb.com">here</a>.
      </p>
    </section>
    </body>
</html>
```


Structure the Browser Uses

Slide Note

# DOM Tree




Web Page Preview

Slide Note

DOM is a standard to model web document as objects

DOM is a way of modelling web documents as objects and manipulating them using a standard programming interface

In the DOM, a document is a collection of objects

# DOM Representation

- DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript.
- A web page is a document that can be either displayed in the browser window or as the HTML source.
- In both cases, it is the same document, but the DOM representation allows it to be manipulated.
- When a web page is opened in the browser, the browser retrieves the page's HTML text and parses it.
- The browser builds up a model of the document's structure and uses this model to draw the page on the screen.
- **DOM represents the page so that programs can change the document structure, style, and content.**

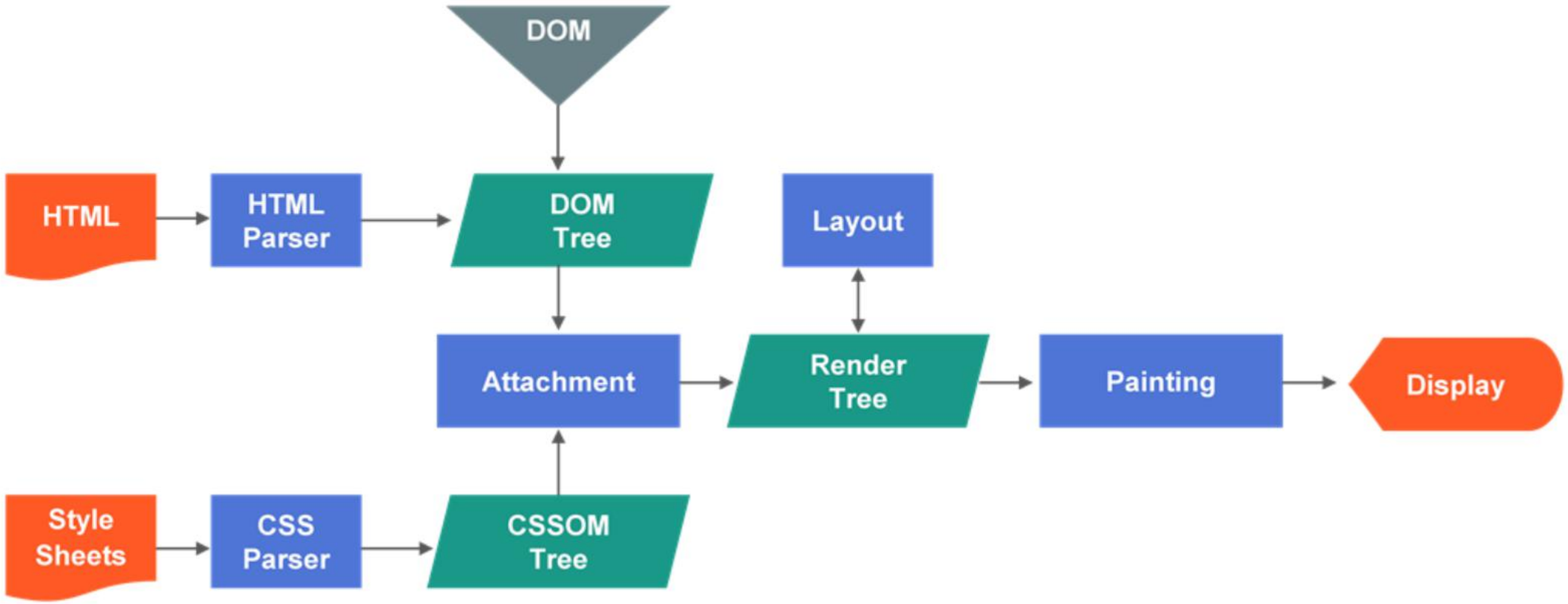
Slide Note

When a web page is loaded, the browser first reads the HTML text and constructs DOM Tree from it.

Then it processes the CSS whether that is inline, embedded, or external CSS and constructs the CSSOM Tree from it.

After these trees are constructed, then it constructs the Render Tree from it.

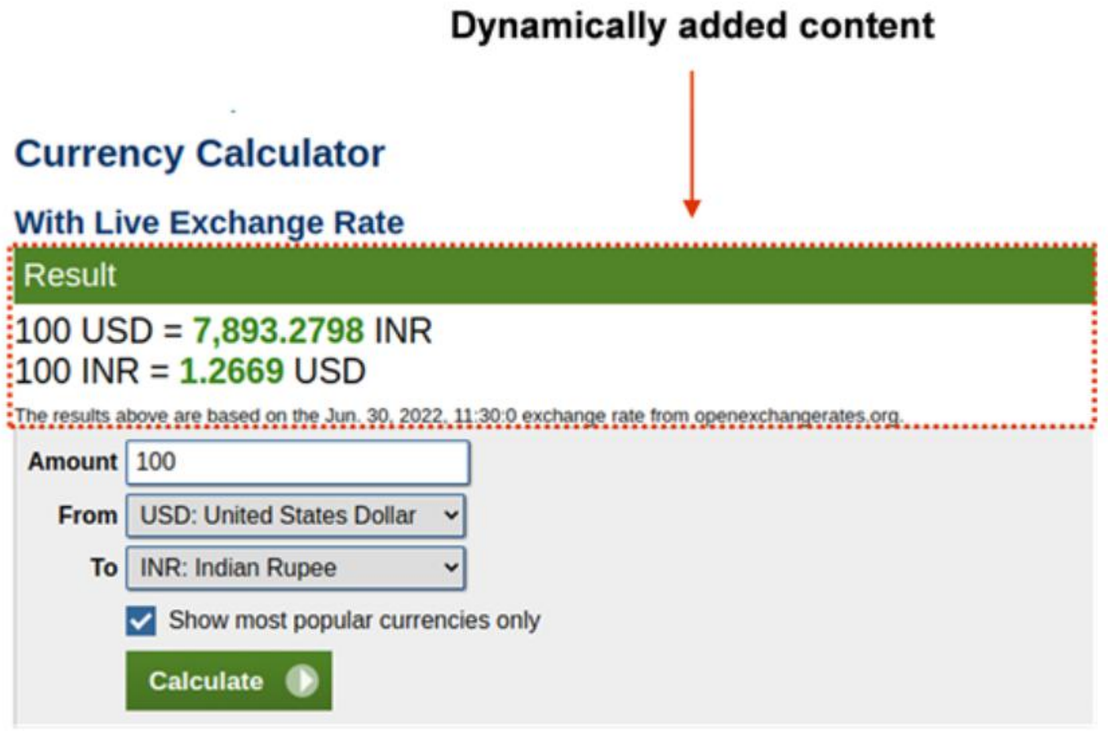
# Sequence of DOM Rendered in Browser



Slide Note

# Document Structure: Currency Converter Page

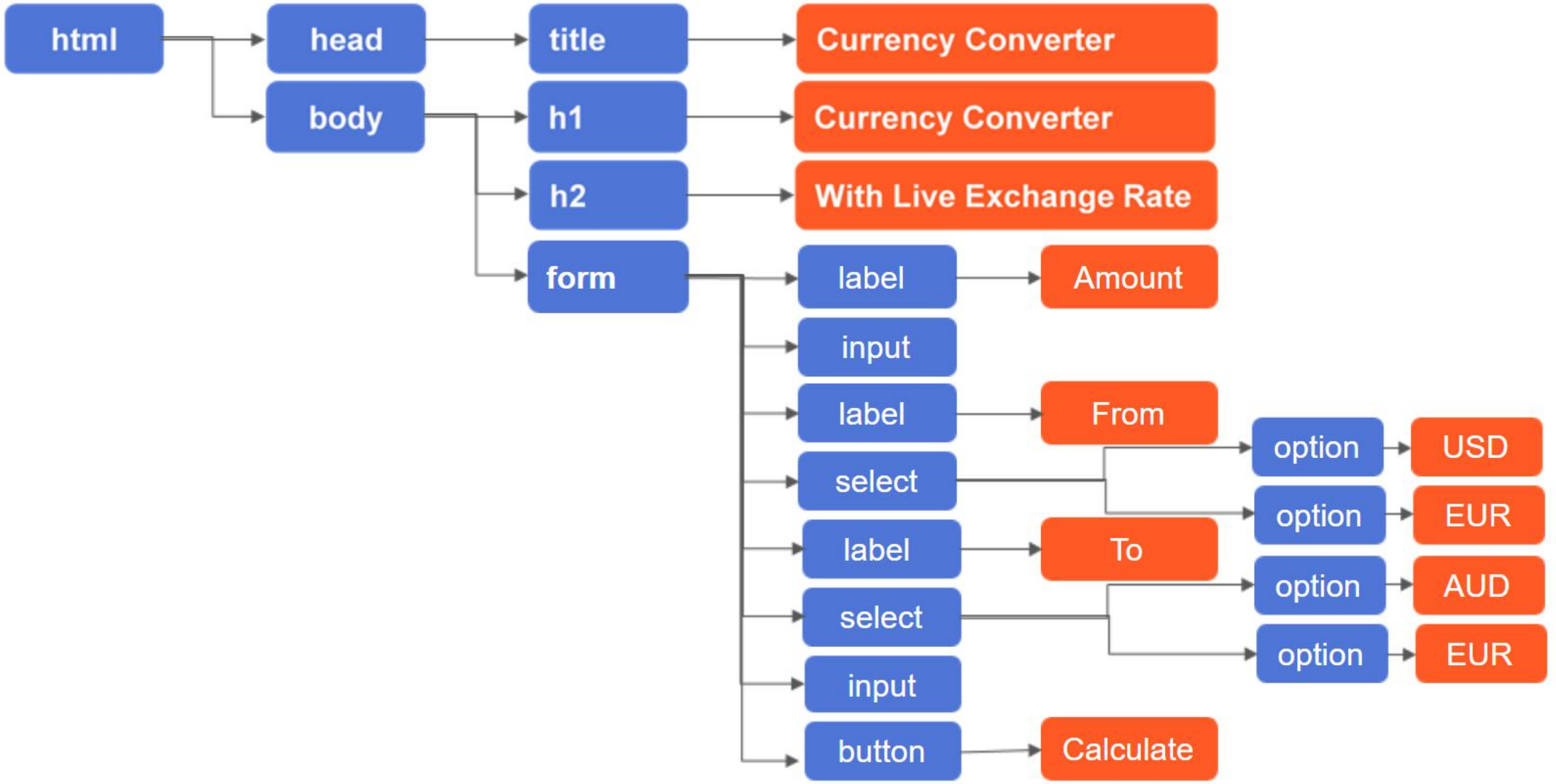
```
<body>
  <h1>Currency Converter</h1>
  <h2>With Live Exchange Rate</h2>
  <form>
    <label for="amount">Amount</label>
    <input type="text" value="100" />
    <label for="from">From</label>
    <select name="fromCurrency" id="">
      <option value="USD">USD: United States
Dollar</option>
      <!--More Options-->
    </select>
    <label for="to">To</label>
    <select name="toCurrency" id="">
      <option value="EUR">EUR: Euro</option>
      <!--More Options-->
    </select>
    <input type="checkbox" name="show" id="">
    Show most popular currencies only
    <button type="submit">Calculate</button>
  </form>
</body>
```




Web Page Preview

Slide Note

# DOM Tree



Slide Note

- Demonstrate the DOM tree created corresponding to the HTML page
- Use the developer tools of the browser
- Demonstrate using an online tool.
- Visit popular websites like bootstrap website and explain the DOM structure from the developer tools on the browser

# Explore DOM

Explore the DOM tree generated for a given web page using the online tool.

Click here for the demo solution.

Click here for the online tool.
Click here to visit bootstrap website.

DEMO

Slide Note

Relation between HTML page and how browser reads, how it containerize and how it displays. How the JS code is running?

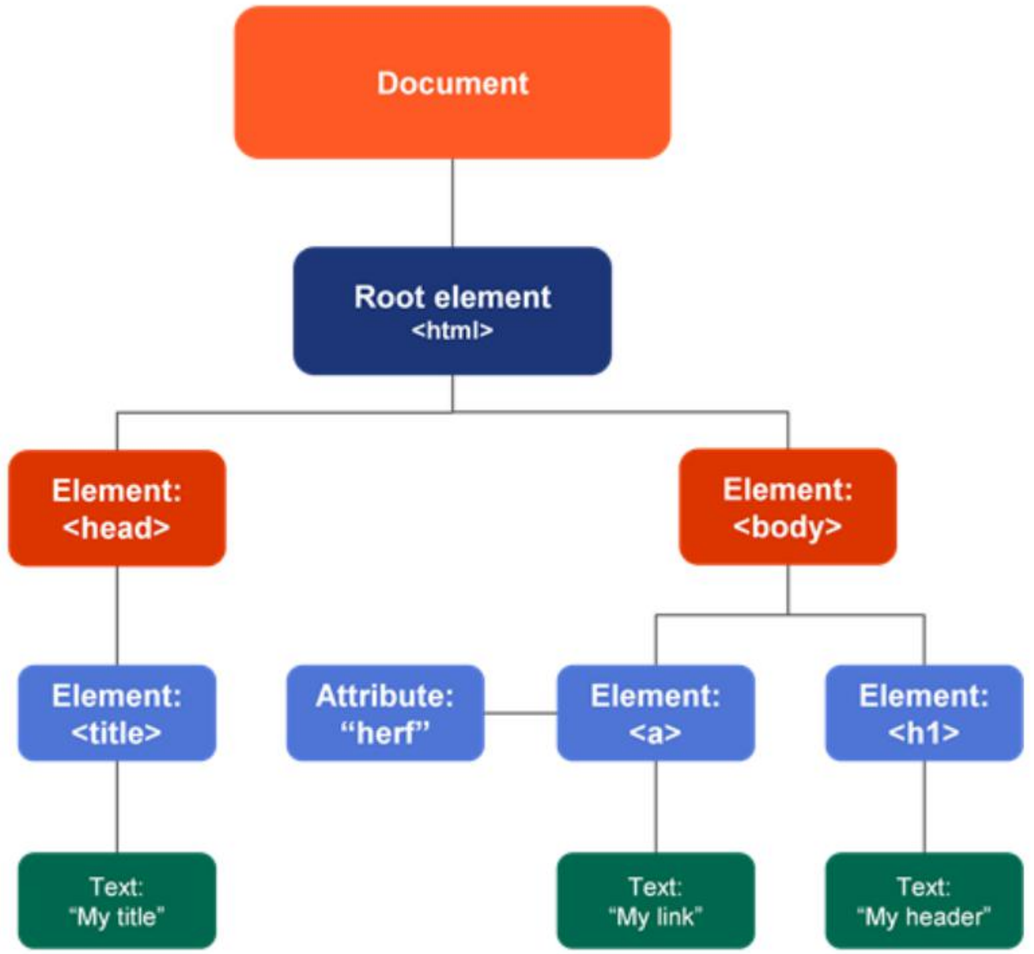
# JavaScript HTML Document Object

- When an HTML document is loaded into a web browser, it becomes a **document object** that serves as an entry point into the web page's content, which is the DOM tree.
- The HTML DOM document object is the owner of all other objects in your web page.
- Any element in the HTML page can be accessed with the document object.
- Some examples where a document object is used to access and manipulate HTML are given below:
  - Finding HTML elements
  - Changing HTML elements
  - Adding and deleting elements
  - Adding or modifying the style properties of existing elements

Slide Note

# Traversing DOM Tree

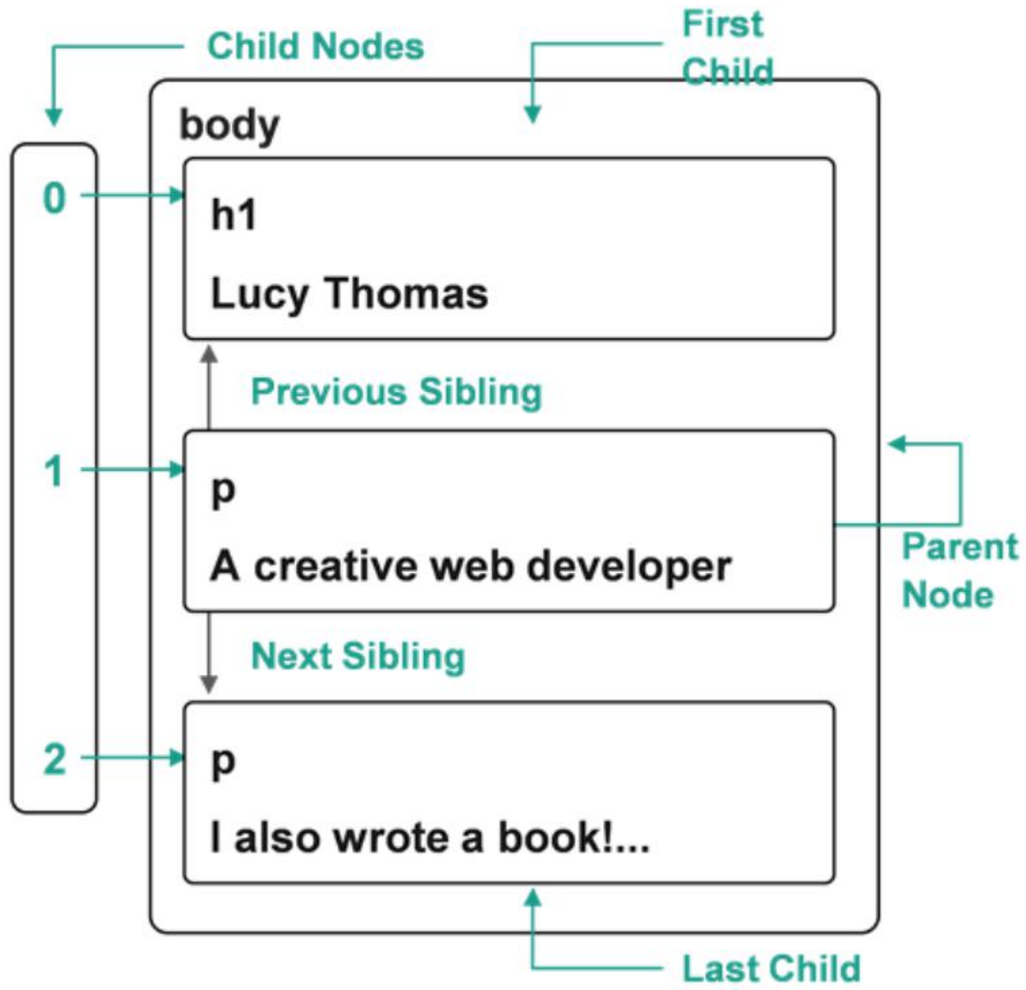
- The DOM tree consists of a tree of objects called nodes.
- The entire document is a document node.
- There are three main type of nodes:
  - Element Node: Any HTML element in the DOM.
  - Text Node: Any lone text outside of a document.
  - Comment Node: An HTML comment.
- You can traverse the node tree using the node relationships with the HTML DOM.



Slide Note

# Node Relationships

- A hierarchical relationship exists among nodes in a node tree.
- The terms parent, child, and sibling are used to describe the relationships.
- The top node is called the root node in the node tree.
- Except for the root (which has no root), every node has exactly one parent.
- A node can have any number of children.
- Nodes with the same parent are called siblings.
- In the example shown, <body> is the parent node and <h1>, <p> and <p> are the three child nodes.



Slide Note

- The **new operator** lets developers create an instance of a user-defined object type or of one of the built-in object types that has a constructor function.
- Refer to this link to know more about new operator.

```
<html>
<head></head>
<body>
  <script>
    let date = new Date();
    document.body.innerHTML = `<h1>
Today's date is ${date} </h1>`;
  </script>
</body>
</html>
```

```
<html>
<head>
  <title>Document</title>
  <script src="script.js"></script>
</head>
<body>
</body>
</html>
----------------------------------------
script.js
let date = new Date();
document.body.innerHTML = `<h1> Today's
date is ${date} </h1>`;
```

# How Do You Add JavaScript to HTML?

- JavaScript code can be added to an HTML document using `<script>` tag.
- The `<script>` tag can be placed in the head section of your HTML or in the body section, depending on when you want the JavaScript to load.
- Adding JavaScript code between the `<head>` tags signals the browser to run the script before loading the rest of the page.
- Add JavaScript code between <body> tags if the HTML content displayed on the page must be modified at runtime.
- To accommodate larger scripts, JavaScript code lives in separate "js" files that can be referenced within HTML documents using the "src" attribute of the <script> tag.

Slide Note

# Dictionary Application

- Consider a scenario where you want to find the synonym of a word.
- Open any popular dictionary website and start typing the word in the search text box and click the search button.
- As shown in the second image, you can see the meaning of the word getting added as a new text content below it.
- The DOM associated with the web page should be manipulated to dynamically add such content.
- **How do you manipulate the DOM nodes?**



**Dynamically added content in the page**

Slide Note

# Find HTML DOM Elements

- To manipulate HTML elements, you must find the elements first.
- HTML elements can be found using several methods:
  - **By id:** If the element is found, the `getElementById()` method will return the element as an object or otherwise returns null.
  - **By tag name:** `getElementsByTagName()` returns the list of all elements matching the tag name as HTMLCollection object
  - **By class name:** `getElementsByClassName()` returns the list of all elements found with the same class name as HTMLCollection object
  - **By CSS selectors:** `querySelectorAll()` returns the list of all elements matching the specified CSS selector( (id, class names, types, attributes, values of attributes)

Slide Note

# Add/Remove Elements From DOM

1. Find HTML elements using various DOM methods.

2. Given a web document, access the HTML elements in the DOM to do various manipulations like adding, removing and updating DOM elements.

Click here for the demo solution.

DEMO

Slide Note

```
<html>
<body>
  <p id="para">Hello World!</p>
  <script>
    document.getElementById("para").
innerHTML = "Welcome to JavaScript
DOM!";
  </script>
</body>
</html>
```

# Change HTML Content Dynamically

- The content of an HTML element can be modified using `innerHTML` property.
- The HTML document in the code contains a <p> element with id="para“.
- You can use the HTML DOM to find the element with id="para" using the getElementById() method.
- Using the innerHTML property, the content of the paragraph is changed to "Welcome to JavaScript DOM“.

Slide Note

# Create a New Element (Node)

- To add a new element to the HTML DOM, the element node should be created before appending it to the existing element.
- In the code, first creates a new `<p>` element using `createElement()`.
- Then text node is created to add text to `<p>` element using `createTextNode()`.
- Then text node is appended to `<p>` element using `appenChild()` method.
- Finally, the new element is added to the existing element using `appendChild()`.
- Note: `insertBefore(newNode, referenceNode)` inserts a node before a *reference node* as a child of a specified *parent node*.

```
<div id="div1">
  <h1 id="h1">Learn HTML DOM.</p>
    <p id="p1">Manipulate DOM by
adding contents dynamically.</p>
</div>
<script>
  const p =
document.createElement("p");
  const node =
document.createTextNode("New paragraph
added.");
  p.appendChild(node);
  const element =
document.getElementById("div1");
  element.appendChild(para);
</script>
```

Slide Note

Event handling is covered in the following slides.

```
<div id="div1">
  <h1 id="h1">Learn HTML DOM.</h1>
  <p id="p1">Manipulate DOM by
adding contents dynamically.</p>
</div>
<script>
  const element =
document.getElementById("p1");
element.remove();
</script>
```

```
<div id="div1">
  <h1 id="h1">Learn HTML DOM.</h1>
  <p id="p1">Manipulate DOM by
adding contents dynamically.</p>
</div>
<script>
  const parent =
document.getElementById("div1");
  const child =
document.getElementById("p1");
  parent.removeChild(child);
</script>
```

# Remove Existing Elements

- Existing elements can be removed using the remove() method.
- In the first code, the HTML document contains a `<div>` element with two child nodes.(`<h1>`, `<p>`).
- Find the `<p>` element to remove `getElementById()` method.
- Finally, execute the `remove()` method on that element.
- In the second code, the child node is removed from the DOM tree using the `removeChild()` method.
- Existing elements can be replaced to the HTML DOM using the `replaceChild()` method.
- Usually, you add or remove elements due to some user actions through events.

Slide Note

# Modify HTML Element's Style

- The HTML DOM allows JavaScript to modify the style of HTML elements using the element `style` property.
- The `style` property only returns style attributes that have been set inline.
- In this code, it returns the string '`color:red`'.
- As shown in the code, the `<p>` element's style property is accessed directly using its "para" id value. Eg., `para.style.color`.
- The font color is later set to a new value 'green' using the same `style` property that overrides any styles that have been set elsewhere.

```
<html>
<body>
  <p id="para"
style="color:red;">Hello World!</p>
  <script>
    //Returns red
    console.log(para.style.color);
    //Change the font color to
green
    para.style.color = "green";
  </script>
</body>
</html>
```

Slide Note

# Modify CSS Properties Dynamically

Manipulate the web page by creating new elements, changing their styles, and adding those new elements to the main document.

Click here for the demo solution.

DEMO

Slide Note

# Modify Attributes of HTML Elements

In JavaScript, there are four methods, that can be used to work with element attributes:

| Method | Description | Example |
|---|---|---|
| hasAttribute() | Returns a true or false Boolean | element.hasAttribute('src'); |
| getAttribute() | Returns the value of a specified attribute or null | element.getAttribute('src'); |
| setAttribute() | Adds or updates value of a specified attribute | element.setAttribute('src', 'img.jpg'); |
| removeAttribute() | Removes an attribute from an element | element.removeAttribute('src'); |

```
// Assign image element
const img = document.querySelector('img');

img.hasAttribute('src'); //returns true
img.getAttribute('src');  //returns "img.png"
img.removeAttribute('src');//remove the src attribute and
value
```

Slide Note

# JavaScript Events

- Events are actions or occurrences that happen in the system you are programming, which the system tells you about so your code can react to them.
- In the case of the Web, events are fired inside the browser window and tend to be attached to a specific item that resides in it.
- Here are examples of different types of events:
  - A web page finishes loading.
  - The user resizes or closes the browser window.
  - The user selects a certain element or hovers the cursor over a certain element.
  - The user chooses a key on the keyboard.
  - A form is submitted.
  - An error occurs.
- Programmers can create *event handler* code that will run when an event fires, allowing web pages to respond appropriately to change.

Slide Note

```
<p>Click this document to activate
the handler.</p>
<script>
  window.addEventListener("click",
() => {
  console.log("You knocked?");
  });
</script>
```

# Event Handling

- Browsers allow us to register functions as handlers for specific events.
- They help in notifying our code when a specific event occurs.
- To react to an event, you can attach an event handler to it, which is called as registering an event handler/event listener.
- An event handler is a block of code that runs when the event fires.
- In this code, `window` is a built-in object representing the browser window that contains the document.
- Calling its `addEventListener` method registers the second argument to be called whenever the event described by its first argument occurs (`click event` in this case).

Slide Note

# Event Handling

1. Handling onload and onclick events and manipulating the DOM.
2. Implement a counter using a buttons click event. Provide an increment button for incrementing the count and a decrement button to decrement the count by 1.

See the Demo here.

DEMO

Slide Note

# Event Handling: Example

- In this code, you store a reference to the button inside a constant called `btn`.
- On loading the page, you are changing the background color of the document to black as the `window.onload` event occurs.
- Next, the `<button>` element has an event called 'click' that fires when the user clicks the button.
- When the button is clicked, the background color of the entire document is changed to a random color generated as the `btn.onclick` event fires.
- When the mouse hovers on the button, the background color of the button turns dark grey as the `btn.onmouseover` event fires.

```
<button>Change color</button>
// JavaScript
   <script>
      const btn = document.querySelector('but
ton')
      function random(number) {
        return Math.floor(Math.random() * (nu
mber+1));
      }
      btn.onclick = function() {
        document.body.style.backgroundColor =
 'rgb(' + random(255) + ',' + random(255) + '
,' + random(255) + ')';
      }
      window.onload = function() {
          document.body.style.backgroundColor
 = 'black';
      }
      btn.onmouseover = function() {
        document.getElementsByTagName("button
")[0].style.backgroundColor = 'darkgrey';
      }
   </script>
```

Slide Note

By convention, JavaScript objects that fire events have a corresponding "onevent" properties (named by prefixing "on" to the name of the event). These properties are called to run the associated handler code when the event is fired and may also be called directly by our code.

In the above code snippet, we have shown how to set a simple greet() function for the click event using the onclick property.

The most flexible way to set an event handler on an element is to use the

EventTarget.addEventListener method.

This approach allows multiple listeners to be assigned to an element, and for listeners to be *removed* if needed using (EventTarget.removeEventListener).

# Registering Event Handlers

## Different Ways of Registering Event Handlers

```
// Using onevent properties

function greet(event){
  // print the event object to console
  console.log('greet:', arguments)
}
btn.onclick = greet;
```

**JavaScript object's onevent (onclick) properties**

```
// Using EventTarget.addEventListener

const btn = document.querySelector('butt
on');
function greet(event){
  // print the event object to console
  console.log('greet:', arguments)
}
btn.addEventListener('click', greet);
```

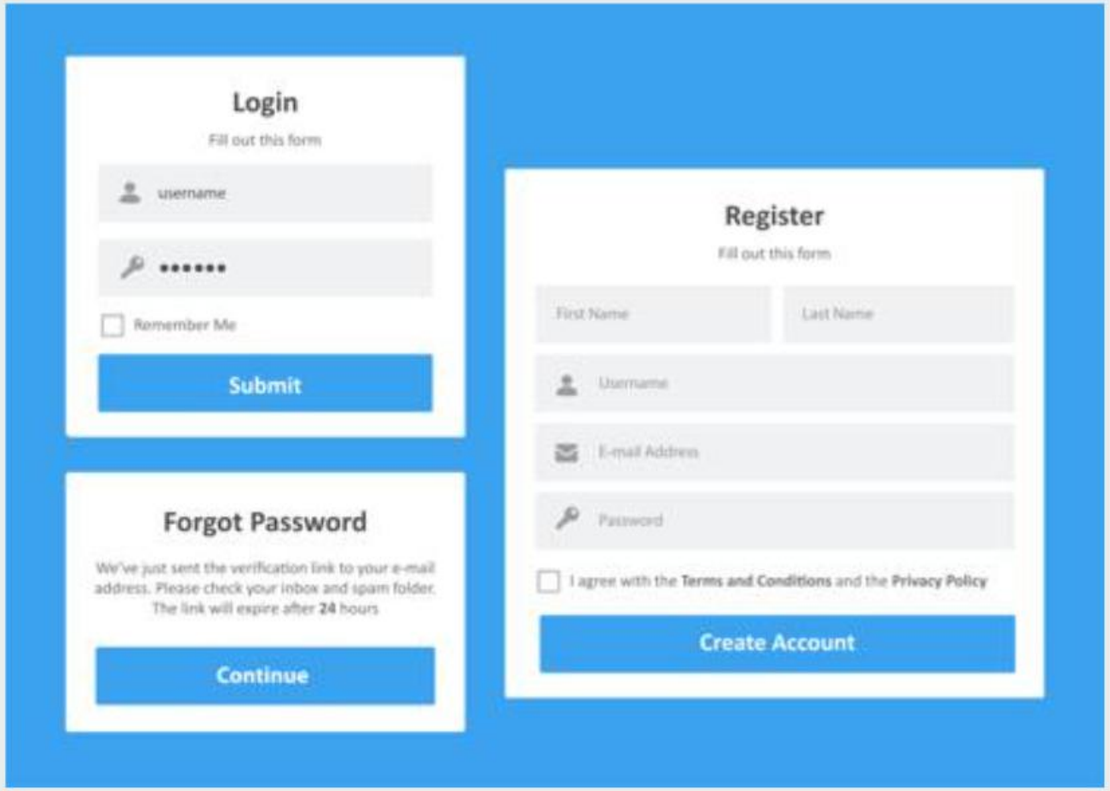
**EventTarget.addEventListener Method**

Slide Note

Direct the thoughts of the learners first towards the point of - What would the user experience be if the Facebook app, at the time of signup, does not validate the inputs provided?

Although the app can still function, would it not create an impact on the trust factor in the minds of users?

Giving an incorrect email, birthdate or not providing critical details will cause problems to the user at some later point of time. Hence, it will become user's responsibility to ensure it's a valid email. Will this not impact user's experience?




# Sign-up Page

- Imagine if a popular app such as Facebook or Gmail accepts an invalid email or date of birth on the sign-up form. What do you think would happen?
- Would performing checks on these inputs as and when the user enters be a better approach for a good user experience?

Slide Note

1. What if your travel start date is June 16th and the app allows you to book the return journey for date June 11th?

2. Would the users find this odd or funny?

3. Let's say your onward journey date is June 15th at 11:00pm and the app allows you to book the return journey for June 15th at 12:30 pm?

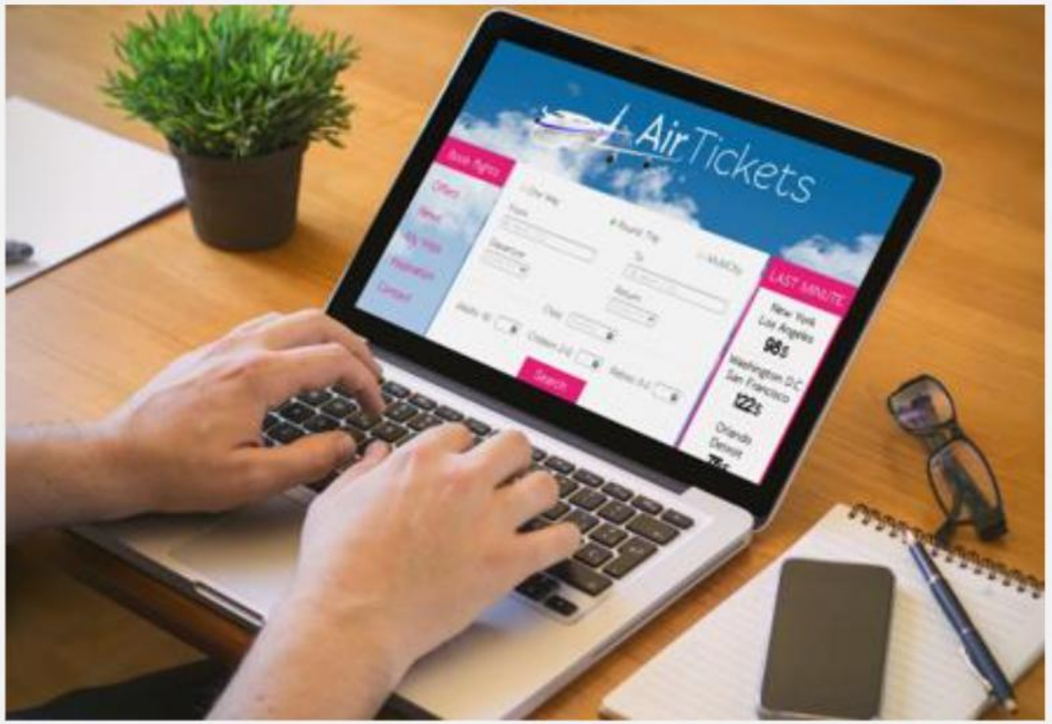
4. Would this not raise a sense of mistrust in the minds of users?

5. Would the user not think twice before using the same app for booking tickets?

6. Instead, would it not provide a better user experience if all the journey dates prior to the current date are disabled?

# Online Ticket Booking

- What will happen if you specify a return date prior to the travel date on a flight booking website?
- Would disabling the dates that are later than the current date help avoid confusion in date selection?
- Do these checks help inspire and more trust to these applications?



Slide Note

# HTML Client-Side Form Validation

- Form validations are a must whenever you accept user inputs to ensure that the data entered is in the correct form and lies within a valid range of data.
- Client-side validation is an initial check and an important feature of a good user experience. By catching invalid data on the client-side, the user can fix it straight away.
- Client-side validation can be performed using:
    - HTML5 built-in functionality: Better performance than JavaScript but cannot be customized.
    - JavaScript: Validation is completely customizable, but code need to be created.
- Some of the typical validation tasks are:
    - Has the user entered valid text in the email field?
    - Has the user filled in all required fields?
    - Has the user entered date in a valid format?

Slide Note

# HTML5 Built-in Form Validation

- Built-in form validation can be performed using validation **attributes** on form elements.
- Some of the common attributes used with form elements are shown here.

| Attributes | Description |
|---|---|
| max | Specifies the maximum value of an input element |
| min | Specifies the minimum value of an input element |
| pattern | Specifies the value pattern of an input element |
| required | Specifies that the input field requires an element |
| disabled | Specifies that the input element should be disabled |
| type | Specifies the type of an input element |
| maxlength | Specifies the maximum number of characters allowed in the input field |
| multiple | Specifies that the user is allowed to enter more than one value in an input field |

Slide Note

# Validate Feedback Form

Create an HTML form with various input fields like username, email, contact No, product, comments and purchase date.
Ensure the form accepts only valid user data before submitting using HTML5 attributes and input types.

Click here for the demo solution.

DEMO

Slide Note

# HTML5 Input Types

- The type attribute of the `<input>` element ensures the user enters a valid data depending on its type.
- Different input types used in HTML are:
  - button
  - checkbox
  - color
  - date
  - email
  - file
  - hidden
  - image
  - month
  - number
  - password
  - radio
  - range
  - reset
  - search
  - submit
  - tel
  - text
  - time
  - url
  - week

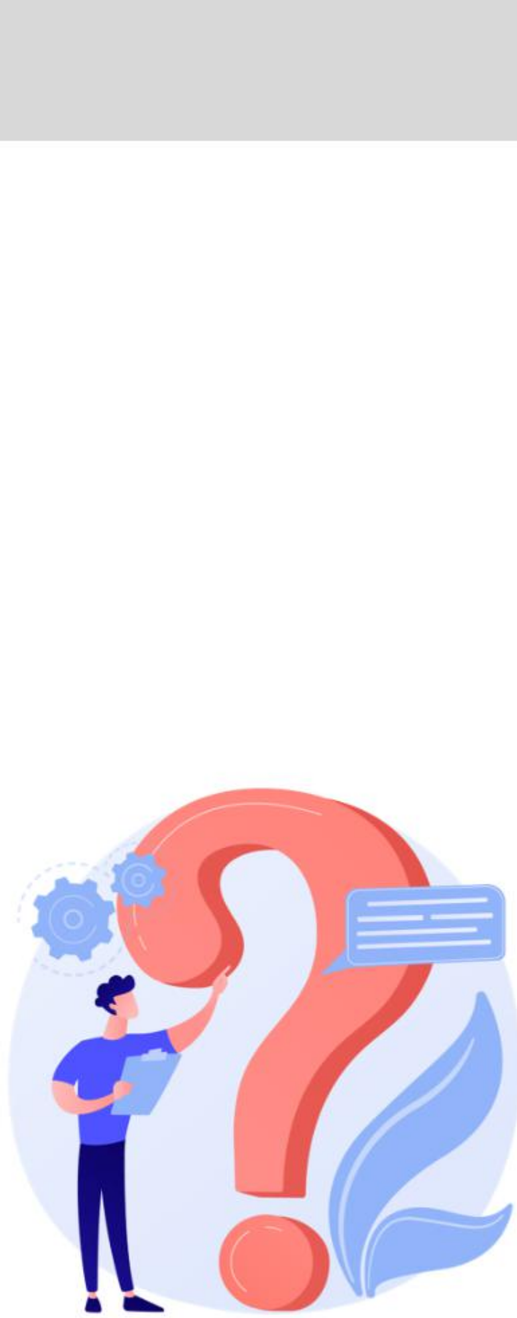
```
<body>
  <form>
    <label
for="name">Username:</label>
    <input type="text" name="name"
id="name" required><br><br>
    <label
for="pass">Password:</label>
    <input type="password" name="pass"
id="pass" required maxlength="10"
minlength="5"><br><br>
    <input type="submit" value="Login"
/>
  </form>
</body>
```

Slide Note

# Quick Check

What is the type of value assigned to the variable element?

```
let element = document.getElementsByTagName( 'button')
```



Slide Note

# Quick Check: Solution

What is the type of value assigned to the variable element?

**HTMLCollection**

There can be multiple elements of the specified tag, so the type of value returned is a collection.