|  |  |
| --- | --- |
| **Question No** | 1 |
| **Skill** | HTML |
| **Proficiency** | Level 2 |
| **Complexity** | Medium |
| **Category** | Implement Concept of Cookies |
| **Question Text** | **Suppose you are building a website for an online bakery, and you want to keep track of the items that a customer adds to their cart.**  **Choose the correct code to retrieve each item in the cart cookie as an individual object with its own properties such as name, quantity, and price? From the option provided below,** |
| **Option A** | var cart = document.cookie.split(";");  for (var i = 0; i < cart.length; i++) {  var item = cart[i];  var eqPos = item.indexOf("=");  var key = eqPos > -1 ? item.substr(0, eqPos) : item;  if (key === "cart") {  var value = eqPos > -1 ? item.substr(eqPos + 1) : "";  // use value to retrieve the items in the cart  }  } |
| **Option B** | var cart = document.cookie;  for (var i = 0; i < cart.length; i++) {  var item = cart[i];  var eqPos = item.indexOf("=");  var key = eqPos > -1 ? item.substr(0, eqPos) : item;  if (key === "cart") {  var value = eqPos > -1 ? item.substr(eqPos + 1) : "";  // use value to retrieve the items in the cart  }  } |
| **Option C** | var cart = JSON.parse(document.cookie);  for (var i = 0; i < cart.length; i++) {  var item = cart[i];  // use item to retrieve the items in the cart  } |
| **Option D** | var cookies = document.cookie.split(";");  for (var i = 0; i < cookies.length; i++) {  var cookie = cookies[i];  var eqPos = cookie.indexOf("=");  var key = eqPos > -1 ? cookie.substr(0, eqPos) : cookie;  if (key === "cart") {  var value = eqPos > -1 ? cookie.substr(eqPos + 1) : "";  var cart = JSON.parse(value) || [];  for (var j = 0; j < cart.length; j++) {  var item = cart[j];  // use item to retrieve the items in the cart  }  }  } |

|  |  |
| --- | --- |
| **Question No** | 2 |
| **Skill** | CSS |
| **Proficiency** | Level 2 |
| **Complexity** | Medium |
| **Category** | Implement Use cases of the preprocessor directives. (SASS, LESS) |
| **Question Text** | **You are building a web page and want to use a CSS preprocessor to simplify your styling. You have written the following code in SCSS:**  $primary-color: #4285f4;  $secondary-color: #ffbb33;  .header {  background-color: $primary-color;  color: $secondary-color;  font-size: 24px;  padding: 10px;  }  **However, when you view the page in your browser, you find that the header has the correct background color and font size, but the text color is not what you expected.**  **Which of the following options is the correct fix for the code?** |
| **Option A** | Replace color: $secondary-color with color: #4285f4. |
| **Option B** | Replace background-color: $primary-color with background-color: #ffbb33. |
| **Option C** | Replace color: $secondary-color with color: #ffbb33. |
| **Option D** | Replace background-color: $primary-color with background-color: #4285f4. |

|  |  |
| --- | --- |
| **Question No** | 3 |
| **Skill** | CSS |
| **Proficiency** | Level 2 |
| **Complexity** | Medium |
| **Category** | Implement Use cases of the preprocessor directives. (SASS, LESS) |
| **Question Text** | **You are building a responsive website and want to simplify your styling by using a CSS preprocessor. You have decided to use LESS. You want to create a mixin that sets the font size, line height, and font family for a text element. You have written the following code in LESS:**  @base-font-size: 16px;  @base-line-height: 1.5;  @base-font-family: Arial, sans-serif;  .text-style(@size: @base-font-size) {  font-size: @size;  line-height: @base-line-height;  font-family: @base-font-family;  }  **You want to use the mixin in your stylesheet and set the font size to 14px. Which of the following options correctly implements the mixin?** |
| **Option A** | ‘.text-style { font-size: 14px; }’ |
| **Option B** | ‘.text-style(14px) { font-size: 14px; }’ |
| **Option C** | ‘.text-style(font-size: 14px) { font-size: 14px; }’ |
| **Option D** | ‘.text-style(14px, font-size) { font-size: 14px; }’ |

|  |  |
| --- | --- |
| **Question no** | 4 |
| **Skill** | CSS |
| **Proficiency** | Level 2 |
| **Complexity** | Medium |
| **Category** | Implement Use cases of the preprocessor directives. (SASS, LESS) |
| **Question text** | **You are using LESS to create a custom grid system for a web page. You want to define a mixin for creating grid columns. You have written the following code in LESS:**  @column-width: 200px;  @gutter-width: 20px;  .grid-column(@index) {  width: (@column-width \* @index) + (@gutter-width \* (@index - 1));  float: left;  }  **You want to create a grid with three columns, each with a width of 200px and 20px of gutters between them. Which of the following options correctly implements the mixin to create the grid columns?** |
| **Option A** | .grid-column(1); .grid-column(2); .grid-column(3); |
| **Option B** | .grid-column(3); .grid-column(2); .grid-column(1); |
| **Option C** | .grid-column(3); .grid-column(3); .grid-column(3); |
| **Option D** | .grid-column(3, 200px, 20px); .grid-column(2, 200px, 20px); .grid-column(1, 200px, 20px); |

|  |  |
| --- | --- |
| **Question No** | 5 |
| **Skill** | HTML |
| **Proficiency** | Level 2 |
| **Complexity** | Medium |
| **Category** | Use accessibility related concepts |
| **Question Text** | **You are creating a blog website, and want to improve its accessibility for users who rely on assistive technologies like screen readers. You want to properly structure the page to clearly convey the content hierarchy.**  **Which of the following options correctly implements semantic elements to structure the header, main content, and footer of the page?** |
| **Option A** | <div>  <div>Header</div>  <div>Main Content</div>  <div>Footer</div>  </div> |
| **Option B** | <div>  <p>Header</p>  <p>Main Content</p>  <p>Footer</p>  </div> |
| **Option C** | <header>Header</header>  <main>Main Content</main>  <footer>Footer</footer> |
| **Option D** | <div id="header">Header</div>  <div id="main">Main Content</div>  <div id="footer">Footer</div> |

|  |  |
| --- | --- |
| **Question No** | 6 |
| **Skill** | HTML |
| **Proficiency** | Level 2 |
| **Complexity** | Medium |
| **Category** | Use accessibility related concepts |
| **Question Text** | **You are creating a form for collecting user information on a website, and you want to ensure that the form is accessible to users with disabilities.**  **Which of the following options correctly implements form controls and labels in HTML to improve accessibility?** |
| **Option A** | <input type="text" value="Name">  <input type="email" value="Email">  <input type="submit" value="Submit"> |
| **Option B** | <form>  <input type="text" placeholder="Name">  <input type="email" placeholder="Email">  <input type="submit" value="Submit">  </form> |
| **Option C** | <form>  <label>Name: <input type="text"></label>  <label>Email: <input type="email"></label>  <input type="submit" value="Submit">  </form> |
| **Option D** | <form>  <div>  <label for="name">Name:</label>  <input type="text" id="name">  </div>  <div>  <label for="email">Email:</label>  <input type="email" id="email">  </div>  <input type="submit" value="Submit">  </form> |

|  |  |
| --- | --- |
| **Question No** | 7 |
| **Skill** | HTML |
| **Proficiency** | Level 2 |
| **Complexity** | Medium |
| **Category** | Explore other HTML5 API'S (Canvas, SVG) |
| **Question Text** | **You are creating a simple drawing application using the Canvas API in HTML. You want to clear the canvas and start a new drawing whenever the user clicks a "Clear" button.**  **Which of the following options correctly implements the "Clear" button using the Canvas API?** |
| **Option A** | <button id="clear">Clear</button>  <canvas id="drawing"></canvas>  <script>  document.getElementById("clear").addEventListener("click", function() {  var canvas = document.getElementById("drawing");  canvas.width = canvas.width;  });  </script> |
| **Option B** | <button id="clear">Clear</button>  <canvas id="drawing"></canvas>  <script>  document.getElementById("clear").addEventListener("click", function() {  var canvas = document.getElementById("drawing");  canvas.clearRect(0, 0, canvas.width, canvas.height);  });  </script> |
| **Option C** | <button id="clear">Clear</button>  <canvas id="drawing"></canvas>  <script>  document.getElementById("clear").addEventListener("click", function() {  var canvas = document.getElementById("drawing");  canvas.fillStyle = "white";  canvas.fillRect(0, 0, canvas.width, canvas.height);  }); |
| **Option D** | <button id="clear">Clear</button>  <canvas id="drawing"></canvas>  <script>  document.getElementById("clear").addEventListener("click", function() {  var canvas = document.getElementById("drawing");  canvas.innerHTML = "";  });  </script> |

|  |  |
| --- | --- |
| **Question No** | 8 |
| **Skill** | JavaScript |
| **Proficiency** | Level 2 |
| **Complexity** | Medium |
| **Category** | Implement Immutability - Array Manipulation |
| **Question Text** | **You are building a web application that allows users to add and remove items from a shopping cart. You want to make sure that the original shopping cart array is not modified when items are added or removed.**  **Which of the following options correctly implements the add and remove item functions using immutability principles in JavaScript?** |
| **Option A** | let shoppingCart = [];  function addItem(item) {  shoppingCart.push(item);  }  function removeItem(item) {  shoppingCart.splice(shoppingCart.indexOf(item), 1);  } |
| **Option B** | let shoppingCart = [];  function addItem(item) {  let newShoppingCart = [...shoppingCart];  newShoppingCart.push(item);  shoppingCart = newShoppingCart;  }  function removeItem(item) {  let newShoppingCart = [...shoppingCart];  newShoppingCart.splice(newShoppingCart.indexOf(item), 1);  shoppingCart = newShoppingCart;  } |
| **Option C** | let shoppingCart = [];  function addItem(item) {  shoppingCart = [...shoppingCart, item];  }  function removeItem(item) {  shoppingCart = shoppingCart.filter(i => i !== item);  } |
| **Option D** | let shoppingCart = [];  function addItem(item) {  shoppingCart.concat(item);  }  function removeItem(item) {  shoppingCart = shoppingCart.slice(0, shoppingCart.indexOf(item)).concat(shoppingCart.slice(shoppingCart.indexOf(item) + 1));  } |

|  |  |
| --- | --- |
| **Question No** | 9 |
| **Skill** | JavaScript |
| **Proficiency** | Level 2 |
| **Complexity** | Simple |
| **Category** | Implement exception handling |
| **Question Text** | **Imagine you are building a JavaScript application that allows users to search for books by title.**  **You want to ensure that the code gracefully handles any errors that may occur during the search process**  **Complete the code to handle errors while retrieving books using the Fetch API in JavaScript**  function searchBooks(title) {  fetch(`https://api.example.com/books?title=${title}`)  .then(response => {  // Check if the response is successful  if (!response.ok) {  // If not  \_\_\_\_(`Error: ${response.status}`);  }  // If the response is successful  return response.\_\_\_\_();  })  .then(books => {  // Do something with the books  })  .catch(error => {  // Log the error to the console  console.\_\_\_\_(`An error occurred while retrieving books: ${error}`);  });  }  try {  searchBooks('javascript');  } catch (error) {  console.\_\_\_\_(`An error occurred while searching for books: ${error}`);  } |
| **Blank 1** | throw new Error |
| **Blank 2** | Json |
| **Blank 3** | Error |
| **Blank 4** | Error |

|  |  |
| --- | --- |
| **Question No** | 10 |
| **Skill** | Javascript |
| **Proficiency** | Level 2 |
| **Complexity** | Medium |
| **Category** | Implement HTML using the DOM - Document Object Model. |
| **Question Text** | **You are building a web application that allows users to change the text of a heading on a page. You have written the following code to select the heading and update its text, but it is not working as expected:**  <h1 id="myHeading">Hello World!</h1>  <button id="myButton">Click me</button>  <script>  let heading = document.getElementById("myHeading");  let button = document.getElementById("myButton");  button.addEventListener("click", function() {  heading.innerHTML = "Goodbye World!";  });  </script>  Which of the following options is the correct code to make the heading text update when the button is clicked? |
| **Option A** | let heading = document.getElementById("myHeading");  let button = document.getElementById("myButton");  button.addEventListener("click", function() {  heading.textContent = "Goodbye World!";  }); |
| **Option B** | let heading = document.getElementById("myHeading");  let button = document.getElementById("myButton");  button.addEventListener("click", function() {  heading.innerText = "Goodbye World!";  }); |
| **Option C** | let heading = document.querySelector("#myHeading");  let button = document.querySelector("#myButton");  button.addEventListener("click", function() {  heading.innerHTML = "Goodbye World!";  }); |
| **Option D** | let heading = document.querySelector("h1");  let button = document.querySelector("button");  button.addEventListener("click", function() {  heading.innerHTML = "Goodbye World!";  }); |

|  |  |
| --- | --- |
| **Question No** | 11 |
| **Skill** | JavaScript |
| **Proficiency** | Level 2 |
| **Complexity** | Medium |
| **Category** | Implement HTML using the DOM - Document Object Model. |
| **Question Text** | **You have a simple HTML page with a button and a paragraph element. When the button is clicked, you want to change the text of the paragraph to indicate that the button was clicked.**  Given the following code, fill in the blanks to correctly implement the button click event listener:  <button id="myButton">Click Me!</button>  <p id="myParagraph">This is a sample paragraph.</p>  <script>  const button = document.\_\_\_\_("myButton");  const paragraph = document.\_\_\_\_("myParagraph");  button.addEventListener("click", () => {  paragraph.\_\_\_\_ = "The button was clicked!";  });  </script> |
| **Blank 1** | getElementById |
| **Blank 2** | getElementById |
| **Blank 3** | textContent |

|  |  |
| --- | --- |
| **Question No** | 12 |
| **Skill** | TypeScript |
| **Proficiency** | Level 2 |
| **Complexity** | Complex |
| **Category** | Implement essential patterns which are not part of JavaScript example : Decorator |
| **Question Text** | **A software engineer is working on an e-commerce website where they have to implement the decorator design pattern to allow customers to add different types of extras to their orders such as gift wrapping, a personalized message, and next-day delivery.**  **The engineer has created an abstract class Order and two concrete classes ProductOrder and ServiceOrder, which extend Order. They have also created a series of decorator classes for each extra, such as GiftWrap, PersonalizedMessage, and NextDayDelivery.**  **The engineer has the following code:**  abstract class Order {  protected price: number;  public abstract getPrice(): number;  }  class ProductOrder extends Order {  public getPrice(): number {  return this.price;  }  }  class ServiceOrder extends Order {  public getPrice(): number {  return this.price;  }  }  class GiftWrap implements Order {  protected decoratedOrder: Order;  public constructor(decoratedOrder: Order) {  this.decoratedOrder = decoratedOrder;  }  public getPrice(): number {  return this.decoratedOrder.getPrice() + 5;  }  }  class PersonalizedMessage implements Order {  protected decoratedOrder: Order;  public constructor(decoratedOrder: Order) {  this.decoratedOrder = decoratedOrder;  }  public getPrice(): number {  return this.decoratedOrder.getPrice() + 2;  }  }  class NextDayDelivery implements Order {  protected decoratedOrder: Order;  public constructor(decoratedOrder: Order) {  this.decoratedOrder = decoratedOrder;  }  public getPrice(): number {  return this.decoratedOrder.getPrice() + 10;  }  } |
|  |  |
| **Option A** | let order = new ProductOrder();  order = new GiftWrap(order);  order = new PersonalizedMessage(order);  order = new NextDayDelivery(order);  console.log(order.getPrice()); |
| **Option B** | let order = new ServiceOrder();  order = new NextDayDelivery(order);  order = new PersonalizedMessage(order);  order = new GiftWrap(order);  console.log(order.getPrice()); |
| **Option C** | let order = new GiftWrap(new ServiceOrder());  order = new PersonalizedMessage(new NextDayDelivery(order));  console.log(order.getPrice()); |
| **Option D** | let order = new ProductOrder();  order = new NextDayDelivery(new PersonalizedMessage(new GiftWrap(order)));  console.log(order.getPrice()); |

|  |  |
| --- | --- |
| **Question No** | 13 |
| **Skill** | Typescript |
| **Proficiency** | Level 2 |
| **Complexity** | Simple |
| **Category** | Work with different JS libraries or frameworks (Example : React + Typescript scaffolding used as per industry standards) |
| **Question Text** | **You want to create a new React + TypeScript project.**  **You have decided to use create-react-app to scaffold the project and add TypeScript support.**  **Given the following code snippet, fill in the blanks to correctly scaffold a new React + TypeScript project:**  npx create-react-app my-app --\_\_\_\_  cd my-app  npm \_\_\_\_ typescript  npm \_\_\_\_ @types/react @types/react-dom  npm \_\_\_\_ |
| **Blank 1** | --template typescript |
| **Blank 2** | Add |
| **Blank 3** | Install |
| **Blank 4** | Start |

|  |  |
| --- | --- |
| **Question No** | 14 |
| **Skill** | Javascript |
| **Proficiency** | Level 2 |
| **Complexity** | Simple |
| **Category** | Data validation using Javascript |
| **Question Text** | **You are developing a form that allows users to input their personal information, including their name, age, and email address. You need to validate the user's input before submitting the form to ensure that the data is in the correct format.**  Which of the following JavaScript code snippets would you use to check if the user's age is a positive integer? |
| **Option A** | if (age > 0 && age % 1 === 0) {  // valid age  } else {  // invalid age  } |
| **Option B** | if (typeof age === 'number' && age > 0) {  // valid age  } else {  // invalid age  } |
| **Option C** | if (isNaN(age) || age <= 0) {  // invalid age  } else {  // valid age  } |
| **Option D** | if (age.isInteger() && age > 0) {  // valid age  } else {  // invalid age  } |

|  |  |
| --- | --- |
| **Question No** | 15 |
| **Skill** | Javascript |
| **Proficiency** | Level 2 |
| **Complexity** | Simple |
| **Category** | Dynamic web pages and Data validation using Javascript |
| **Question Text** | **You are building a dynamic web page that requires user input in the form of a name and email address.**  **You want to validate the user's input before submitting the form.**  **Given the following HTML form:**  <form id="myForm">  <input type="text" id="name" required>  <input type="email" id="email" required>  <input type="submit" value="Submit">  </form>  And the following JavaScript code that is meant to validate the form data before submission:  const form = document.getElementById("myForm");  form.addEventListener("submit", function(event) {  event.preventDefault();  const name = document.getElementById("name").value;  const email = document.getElementById("email").value;    if (!\_\_\_\_(name)) {  alert("Name is required");  return;  }    if (!\_\_\_\_\_.test(email)) {  alert("Email is invalid");  return;  }    alert("Form submitted successfully");  });  Which of the following will fill in the two blanks in the JavaScript code to properly validate the form data before submission? |
| **Option 1** | isEmpty and /\S+@\S+\.\S+/ |
| **Option 2** | Boolean and /\S+@\S+\.\S+/ |
| **Option 3** | isEmpty and /^\w+([\.-]?\w+)@\w+([\.-]?\w+)(\.\w{2,3})+$/ |
| **Option 4** | Boolean and /^\w+([\.-]?\w+)@\w+([\.-]?\w+)(\.\w{2,3})+$/ |