**Assignment 1**

**Question 1: Answer The Following Questions**

1. **What is Object Destructuring?**

Object destructuring is a syntax in JavaScript is a concise way to extract properties from an object and assign them to variables. Instead of accessing each property individually, you can unpack values directly from the object into variables. This enhances code readability and reduces repetition. It also allows for renaming variables and setting default values when a property is missing.

1. **Explain Closures in JavaScript.**

A closure in JavaScript is when a function retains access to its lexical scope, even after the outer function has executed. This allows the inner function to access variables from the outer function, effectively "remembering" the environment in which it was created. Closures are useful for creating private variables, handling asynchronous code, and functional programming techniques like currying. They are a core concept in JavaScript, enabling more flexible and powerful code.

1. **What do you understand by hoisting in JavaScript?**

Hoisting allows for variable and function declarations to be used before they are declared in the code, but the behavior differs between var, let, const, and function declarations.

|  |  |  |
| --- | --- | --- |
| **Type of Declaration** | **Behavior** | **Example** |
| **var** | Declarations are hoisted to the top of the function or global scope but initialized with undefined. | console.log(a);  // undefined  var a = 5; |
| **let** and **const** | Declarations are hoisted to the top of the block scope but are not initialized. They are in a "temporal dead zone" until the declaration is encountered. | console.log(b);  // ReferenceError  let b = 5; |
| **function** | Function declarations are hoisted with their entire definition. | console.log(foo()); // 'bar'  function foo() {    return "bar";  } |

1. **❌**
2. **What is Eager and Lazy loading?**

* **Eager Loading:** Loads resources or executes code immediately, which can lead to slower initial load times but ensures all data is ready to use when needed.

function loadData() {

  const users = fetch("/api/users") // Fetch all users immediately

    .then((response) => response.json())

    .then((data) => console.log(data));

}

loadData(); // Data is loaded eagerly when the function is called

* **Lazy Loading:** Defers loading until it is necessary, optimizing performance by reducing initial load times and conserving resources, but may introduce delays when accessing deferred resources.

function loadUserData() {

  return fetch("/api/user").then((response) => response.json());

}

document.getElementById("userButton").addEventListener("click", () => {

  loadUserData().then((data) => console.log(data));

  // Data is fetched only when the button is clicked

});

* Both approaches are useful in different scenarios, and often, a combination of eager and lazy loading strategies is employed in JavaScript applications to balance performance and user experience.

1. **❌**
2. **How do you specify units in the CSS? What are the different ways to do it?**

|  |  |  |
| --- | --- | --- |
| **Unit Type** | **Unit** | **Description** |
| **Absolute Units** |  |  |
| **Pixels** | px | Represents one pixel on the screen. |
| **Points** | pt | Represents 1/72 of an inch, traditionally used in print media. |
| **Inches** | in | Represents a physical inch. |
| **Centimeters** | cm | Represents a physical centimeter. |
| **Millimeters** | mm | Represents a physical millimeter. |
| **Picas** | pc | 1 pica equals 12 points. |
| **Relative Units** |  |  |
| **Percentages** | % | Relative to the parent element’s size. |
| **Ems** | em | Relative to the font-size of the element itself. If the font-size is 16px, then 1em is 16px. |
| **Root ems** | rem | Relative to the font-size of the root element (<html>). If the root font-size is 16px, then 1rem is 16px. |
| **View Width** | vw | 1vw is 1% of the viewport width. |
| **View Height** | vh | 1vh is 1% of the viewport height. |
| **View Min** | vmin | 1vmin is 1% of the smaller dimension (width or height) of the viewport. |
| **View Max** | vmax | 1vmax is 1% of the larger dimension (width or height) of the viewport. |
| **Character (ch)** | ch | Relative to the width of the zero (0) character of the element’s font. |
| **Ex (ex)** | ex | Relative to the height of the lowercase x of the element’s font. |
| **Cap Height** | cap | Relative to the height of the capital letters in the font. |
| **Line Height** | lh | Relative to the line-height of the element. |

1. **What property is used for changing the font face?**

The CSS property used for changing the font face is the font-family property.

1. **How is border-box different from content-box?**

|  |  |  |
| --- | --- | --- |
| **Property** | **content-box (Default)** | **border-box** |
| **Width/Height** | Applies only to the content. | Includes content, padding, and border. |
| **Total Width Calculation** | width + padding + border | width (fixed, includes padding and border) |
| **Total Height Calculation** | height + padding + border | height (fixed, includes padding and border) |
| **Content Area Calculation** | Width/height is calculated excluding padding and border. | Width/height is calculated including padding and border. |
| **Use Cases** | When content size is crucial, and padding or border should be outside. | Easier layout management with fixed total size of the element. |
| **Visual Example** | If width: 200px and padding: 20px, total width = 240px. | If width: 200px and padding: 20px, total width = 200px. |
| **Default Behavior** | Yes, this is the default box-sizing model in CSS. | No, you must explicitly set box-sizing: border-box;. |

1. **How to center align a div inside another div? [2 Ways]**

* **Method 1: Using Flexbox**

.parent {

  display: flex;

  justify-content: center; /\* Centers horizontally \*/

  align-items: center;     /\* Centers vertically \*/

  height: 100vh;           /\* Full viewport height, adjust as needed \*/

}

.child {

  width: 200px;  /\* Set width if needed \*/

  height: 100px; /\* Set height if needed \*/

}

* **Method 2: Using CSS Grid**

.parent {

  display: grid;

  place-items: center; /\* Centers both horizontally and vertically \*/

  height: 100vh;       /\* Full viewport height, adjust as needed \*/

}

.child {

  width: 200px;  /\* Set width if needed \*/

  height: 100px; /\* Set height if needed \*/

}

**Question2: What is The Output?**

1. 6 – 8
2. TypeError: freddie.colorChange is not a function
3. TypeError: member.getFullName is not a function
4. 2 – 4

**Question 3:**

1. **a) What gets logged to the console when the user clicks on “Button 4” and why?**

The value that gets logged to the console will be 5.

* The issue lies in the use of var in the for loop. The variable i is function-scoped, meaning there is only one i shared among all iterations of the loop. By the time any of the buttons are clicked, the loop has already finished executing, and i has been incremented to 5 after the final iteration (since the loop runs while i < 5).
* Therefore, when any button is clicked, the value of i will be the final value it reached, which is 5, and that value will be logged for every button.

b) Provide one or more alternate implementations that will work as expected.

for (let i = 0; i < 5; i++) {

  var btn = document.createElement("button");

  btn.appendChild(document.createTextNode("Button" + i));

  btn.addEventListener("click", function () {

    console.log(i);

  });

  document.body.appendChild(btn);

}

1. **🡪 12 In VS Code**

**Question4: True Or False**

|  |  |
| --- | --- |
| 1. False | 1. ❌ |
| 1. False | 1. False |
| 1. False | 1. False |
| 1. True | 1. False |
| 1. True | 1. False |