**Mock Test1**

**1.**

Static: Static positioning is the default positioning behavior for elements. Elements with static position are positioned according to the normal flow of the document.

Example:

<style>

.box {

width: 200px;

height: 200px;

background-color: red;

}

</style>

<div class="box">Static</div>

Fixed: Fixed positioning allows you to position an element relative to the browser window, regardless of scrolling. The element remains fixed in the specified position even if the page is scrolled.

Example:

<style>

.box {

position: fixed;

top: 20px;

left: 20px;

width: 200px;

height: 200px;

background-color: blue;

}

</style>

<div class="box">Fixed</div>

Sticky: Sticky positioning is a hybrid of relative and fixed positioning. It behaves like a relative positioned element until the user scrolls to a specific threshold, and then it becomes fixed.

Example:

<style>

.box {

position: sticky;

top: 50px;

width: 200px;

height: 200px;

background-color: green;

}

</style>

<div class="box">Sticky</div>

Relative: Relative positioning positions an element relative to its normal position in the document flow. It allows you to adjust an element's position using top, right, bottom, and left properties.

Example:

<style>

.box {

position: relative;

top: 50px;

left: 50px;

width: 200px;

height: 200px;

background-color: orange;

}

</style>

<div class="box">Relative</div>

Absolute: Absolute positioning removes the element from the normal document flow and positions it relative to its nearest positioned ancestor or to the initial containing block.

Example:

<style>

.container {

position: relative;

height: 400px;

}

.box {

position: absolute;

top: 50px;

left: 50px;

width: 200px;

height: 200px;

background-color: yellow;

}

</style>

<div class="container">

<div class="box">Absolute</div>

</div>

2.

Execution context in JavaScript refers to the environment in which JavaScript code is executed. It contains all the necessary information for the interpreter to execute the code, including variable and function declarations, scope chain, and the value of the this keyword.

An execution context can be thought of as a wrapper around the code that provides the context for its execution. Whenever a function is invoked, a new execution context is created, and it consists of three main components:

Variable Environment: The variable environment of an execution context contains all the variables and function declarations defined within the context. It includes function arguments, local variables, and function declarations. The variable environment also maintains the reference to the outer environment or the lexical environment, forming a chain known as the scope chain.

Scope Chain: The scope chain is a linked list of variable environments. It allows the interpreter to resolve variables by traversing through the chain until a matching variable name is found. The scope chain ensures that variables can be accessed from outer environments and provides lexical scoping in JavaScript.

Lexical this: The lexical this value represents the reference to the current execution context. It depends on how a function is invoked. The value of this is determined at runtime based on the context of the function invocation, such as the object it is called upon or the way it is called (e.g., as a method, constructor, or standalone function).