## Branching in JS

Q: What is branching in JavaScript?

A: Branching in JavaScript refers to the ability to control the flow of program execution based on different conditions. It allows the program to make decisions and choose different paths of execution depending on the values of variables or expressions.

Q: What are valid conditions for branching?

A: Valid conditions for branching in JavaScript can be any expression that evaluates to a boolean value (true or false). This includes comparison operators such as ==, ===, !=, !==, <, >, <=, >=, logical operators such as && (AND), || (OR), and ! (NOT), as well as any truthy or falsy values.

Q: What are some examples of when I would use branching?

A: Branching is commonly used in JavaScript to handle various scenarios, such as:

Executing different code blocks based on user input or user interactions.

Validating user input and performing different actions based on the validity of the input.

Implementing error handling and exception handling to handle different error conditions.

Controlling the flow of a program based on the state of application data or external factors.

Q: What problems can arise when branching?

A: Some potential problems that can arise when using branching include:

Overly complex branching logic, which can make code harder to read, understand, and maintain.

The risk of introducing bugs or errors, especially when dealing with nested if statements or complex conditions.

Difficulty in testing, as each branch of the code may need to be tested separately to ensure proper behavior under different conditions.

Q: What are some best practices to use when branching?

A: To ensure cleaner and more maintainable code when using branching, consider the following best practices:

Keep branching logic simple and straightforward, avoiding excessive nesting and complex conditions.

Use meaningful variable names and comments to clearly document the purpose and intention of each branch.

Break down complex branching logic into smaller, more manageable functions or modules.

Consider using guard clauses or early returns to handle exceptional cases at the beginning of a function.

Regularly review and refactor branching logic to improve readability and maintainability.

Q: Does a switch still count as branching?

A: Yes, a switch statement still falls under the category of branching in JavaScript. It provides an alternative way to implement branching logic, often used when there are multiple conditions to evaluate.

Q: Besides if statements and switches, are there any other ways to branch?

A: Yes, besides if statements and switches, other techniques for branching in JavaScript include the ternary operator (?:), guard clauses, and conditional execution with logical operators like && (AND) and || (OR). Each technique offers flexibility in handling different types of conditions and can be chosen based on the specific requirements of the code.

## Events in JS

Q: What are events in JavaScript?

A: Events in JavaScript are actions or occurrences that happen in the browser, triggered by user interactions (like clicking a button), browser actions (like finishing loading a page), or other sources.

Q: How do they work? Does the code sit and wait until the condition is met?

A: Events in JavaScript work asynchronously. The code doesn't sit and wait; instead, it registers event listeners that wait for specific events to occur. When an event occurs, the corresponding event listener function is executed.

Q: How do you respond to events from the browser?

A: You respond to events from the browser by attaching event listeners to DOM elements. These listeners "listen" for specific events, like clicks or keypresses, and execute a function when the event occurs.

Q: What are the possible parameters for addEventListener?

A: The possible parameters for addEventListener include the event type (e.g., "click", "mouseover"), the event handler function, and an optional third parameter to specify options like whether the event should be captured during the capturing phase.

Q: Along with "click", "mouseover", and "keydown", what other options are there?

A: Other options for event types include "mouseenter", "mouseleave", "keyup", "submit", "focus", "blur", and many more. These correspond to various user interactions and browser actions.

Q: What are the capturing phase or the bubbling phase?

A: The capturing and bubbling phases are two ways events propagate through the DOM. In the capturing phase, the event starts from the outermost element and propagates down to the target element. In the bubbling phase, the event starts at the target element and propagates up to the outermost element.

Q: Explain the phases to me as if I were 10.

A: Sure! Imagine you have a bunch of nesting dolls stacked inside each other. When you tap the outermost doll, the tap travels inside through each doll until it reaches the smallest one (capturing phase). Then, it travels back out from the smallest doll to the outermost one (bubbling phase).

Q: Why does it matter where the event starts from? Won't the called function execute no matter where the event is in the DOM?

A: It matters because knowing where the event starts from helps control when and where event listeners are triggered. Depending on the phase (capturing or bubbling), you can choose where to intercept the event and execute your code accordingly. This allows for more precise event handling in complex web applications.