1. What is the difference between the following 2 statements?

setTimeout(booyah, 2000);

setTimeout(booyah(), 2000);

Answer :

The difference between the two statements is that the first statement **setTimeout(booyah, 2000)** sets a delay of 2000 milliseconds (2 seconds) and then executes the **booyah** function after the specified delay.

The second statement **setTimeout(booyah(), 2000)** immediately calls the **booyah** function and passes its return value as the first argument to **setTimeout**. This means that the **booyah** function will execute immediately and not after the specified delay.

To clarify, if you want to execute the **booyah** function after a delay of 2000 milliseconds, you should use the first statement (**setTimeout(booyah, 2000)**), but if you want to immediately execute the **booyah** function and pass its return value to **setTimeout**, you should use the second statement (**setTimeout(booyah(), 2000)**).

1. What do the following 2 alerts display (answer without running the code)?

var myfunc = function(a, x) {

return a \* x;

};

var x = myfunc(2, 3);

var y = myfunc;

alert(x);

alert(y(2,3));

Ans:

The first alert will display the value of **x**, which is the result of calling **myfunc(2, 3)**. Since **myfunc** multiplies its first argument **a** (which is 2) with its second argument **x** (which is 3), **x** will be 6.

The second alert will also display 6 because **y** is assigned the function **myfunc**, so calling **y(2, 3)** is equivalent to calling **myfunc(2, 3)**, which returns 6.

3. Write functions booyah1 and booyah2 so that in both cases below, an alert box comes up after 2 seconds that says “BOOYAH!” setTimeout(booyah1, 2000); setTimeout(booyah2(), 2000);

Ans:

To make an alert box that says "BOOYAH!" appear after 2 seconds using **setTimeout**, you can define the following two functions:

1. **booyah1** function as a named function that does not take any arguments and simply calls the **alert** function with the message "BOOYAH!":

function booyah1() {

alert("BOOYAH!");

}

1. **booyah2** function as an anonymous function expression that also does not take any arguments and calls the **alert** function with the message "BOOYAH!":

var booyah2 = function() {

alert("BOOYAH!");

};

Then, you can call **setTimeout** with each of these functions as the first argument and the time delay of 2000 milliseconds as the second argument:

setTimeout(booyah1, 2000);

setTimeout(booyah2, 2000);

Note that when calling **setTimeout** with **booyah2**, you should not include the parentheses after the function name, because you want to pass the function itself as an argument to **setTimeout**, not the result of calling the function.

Top of Form

1. What is "Unobtrusive Javascript"? What is the practical application of Unobtrusive Javascript (and the reasons for using it)?

Ans:

Unobtrusive JavaScript refers to the practice of using JavaScript code in a way that does not interfere with the normal functioning of a web page, particularly in cases where a user does not have JavaScript enabled in their browser.

The main reason for using unobtrusive JavaScript is to improve the accessibility and usability of web pages. By separating JavaScript code from HTML markup and using a more modular approach to programming, developers can create more maintainable and easily extendable code. This approach also ensures that users who have JavaScript disabled or are using devices that do not support JavaScript can still access the content and functionality of the web page.

Some practical applications of unobtrusive JavaScript include:

1. Form validation: Instead of using inline JavaScript code to validate user input, developers can use unobtrusive JavaScript to validate form data after submission. This approach ensures that users who do not have JavaScript enabled can still submit forms, and provides a more seamless user experience for those who do.
2. Dynamic page updates: Unobtrusive JavaScript can be used to update parts of a web page without requiring a full page refresh. This approach can be particularly useful in cases where users are interacting with dynamic content, such as social media feeds or search results.
3. Event handling: Instead of using inline JavaScript code to handle events such as clicks or mouseovers, developers can use unobtrusive JavaScript to attach event listeners to elements on a web page. This approach provides a more modular and maintainable way of handling events, and ensures that users who do not have JavaScript enabled can still interact with the web page.