UNIT 03

LESSON 03.06

Hoisting

Function Expressions

Anonymous Functions

hoisting

Hoisting refers to functions being lifted to the top of their scope, so that they are available everywhere in their scope. Since they are hoisted, functions can be called before their declaration appears in the code.

1. Declare a function and call it before and after:

```
console.log(1, sayHello('Joe')); // Hello, Joe!

function sayHello(name) {
    return **Hello, ${name}!**;
}

console.log(2, sayHello('Joe')); // Hello, Joe!
```

Hoisting does not work for variables, however. To access a variable, it must have already been declared:

2. Try to use a variable before it is declared. You get an error:

```
console.log(`Do you like ${froot}?`);
// ReferenceError: froot is not defined
let froot = 'apples';
console.log(`Do you like ${froot}?`);
// Do you like apples?
```

function experession

A function expression is a variable with a function for a value. As a variable, it doesn't get hoisted. A function expression must be defined *before* it can be called.

3. Write a function expression:

```
    const welcomeUser = function(user) {
        return `Welcome ${user}!`;
    }
```

4. Call the function above itself. We get an error:

```
    console.log(welcomeUser('Jane1'));
    // ReferenceError: welcomeUser is not defined

    const welcomeUser = function(user) {
        return `*Welcome ${user}!`;
    }

    console.log(welcomeUser('Jane1'));
```

5. Comment out the error line and call the function down below. It works:

```
    // console.log(welcomeUser('Jane123'));
    // ReferenceError: welcomeUser is not defined

    const welcomeUser = function(user) {
        return **Welcome ${user}**;
    }

    console.log(welcomeUser('Janet456'));
    // Welcome Janet456
```

anonymous functions

An anonymous function is a function that does not have a name. Anytime you see the word function followed by parentheses with no name in between, it's an anonymous function:

```
    // named function:
    function addNumbers(x, y) {
        return x y;
    }

    // anonymous function:
    function(x, y) {
        return x, y;
    }
```

The reason functions have names at all is so they can be called. It stands to reason then that anonymous functions do not need to be called. But if that's true, how do they run?

function set equal to an event property

An anonymous function may be set equal to the event property of an object. When that event happens, the function runs. First let's just look at a function that is called by a listener:

6. Get the BTN 1 button and have it call a function when clicked:

```
    const btn1 = document.getElementById('btn-1');
    btn1.addEventListener('click', doSomething);
```

7. Define the function:

```
    function doSomething() {
        output.textContent = 'You clicked the button! That did
something!';
    }
```

8. Get the element where the function output goes:

```
    const output = document.getElementById('output');
```

9. Reload the page and click the button. We should get the message: *You clicked the button! That did something!*

This review of addEventListener is just to set up the contrast between calling function with a name and running an anonymous function on click.

10. Get BTN 2:

```
    const btn2 = document.getElementById('btn-2');
```

onclick property

Events that can be called on objects co-exist as properties. The 'click' event has its counterpart in the **onclick** property, which is called on the button and set equal to a function. When the click takes place, the function runs:

11. Access the button's onclick property and set it equal to an anonymous function:

```
    btn2.onclick = function() {
        output.textContent = 'Hello from the anonymous function!';
    }
```

For that matter, an anonymous function can be used in a listener, rather than callng a function.

inline anonymous functions

Anonymous functions do not have to come to the right of an equal sign. An anonymous function can be written inline, such as inside the addEventListener method.

13. Get BTN 3 and have it listen for a click. Instead of calling a function by name, as we usually do, just run an inline anonymous function, right there on the spot:

```
    const btn3 = document.getElementById('btn-3');

    btn3.addEventListener('click', function() {
        output.textContent = 'Hello from inline anonymous function!';
    })
```

Let's try one more example of a listener that calls a function by name vs. a listener that runs an anonymous function, inline:

14. Have the body listen for the 'load' event, which fires when the body, is fully loaded, and to call a function when the loading of the page is complete:

```
    document.body.addEventListener('load', onBodyLoaded);

    function onBodyLoaded() {
        output4.textContent = 'BODY LOADED says onBodyLoaded function';
    }
```

15. Refactor the listener to run an inline anonymous function instead of calling a function by name:

```
    document.body.addEventListener('load', function() {
        output5.textContent = 'BODY LOADED says inline anonymous 
function';
    });
```

16. Do away with the listener altogether, and instead use the onload event property to run the anonymous function:

```
    document.body.onload = function() {
        output6.textContent = 'BODY LOADED says document.body.onload = 
anonymous function';
    };
```

keyboard object and events

The pressing of any key is also an event. These are typically also used in conjunction with anonymous functions.

The **document** can listen for a keyboard event and call a function when any key is pressed (**keydown**) or released (**keyup**).

The event properties for the keyboard are **event.key** which is the key that was pressed and **event.keyCode** which is the key's numeric code. The function takes **event** as its argument.

17. Have the document listen for a **keydown** event and call a function when any key is pressed. The function reports the **key** and its **keyCode**.

```
    document.addEventListener('keydown', reportKeyAndCode);

    function reportKeyAndCode(event) {
        output4.textContent = `listener keydown calls reportKeyAndCode
function:\nKey: ${event.key} Code: ${event.keyCode}`;
    }
```

18. Refactor the above to use an inline anonymous function:

```
    document.addEventListener('keydown', function(event) {
        output5.textContent = `listener keydown runs anonymous inline
function:\nKey: ${event.key} Code: ${event.keyCode}`;
    });
```

19. And finally, use the onkeydown property to call an anonymous function:

```
    document.onkeyup = function(event) {
        output6.textContent = `onkeyup = anonymous function \nKey:
${event.key} Code: ${event.keyCode}`;
    };
```

- **END: Lesson 03.06**
- **NEXT: Lesson 03.07**