Lesson 03.05

- Hoisting: Function Declarations vs. Expressions
- Anonymous Functions
- Inline Anonymous Callback Functions
- onclick property of DOM objects

hoisting

- Hoisting refers to functions being lifted automitically to the top of their scope.
- Hoisting lets functions be called from lines above where they are declared.
- Hoisting does not apply to variables--only functions.
- To use a variable, it must have already been declared.

1. Declare a variable, fruit, and try to log it from the line above:

```
console.log(fruit); // Error:  Cannot access 'fruit' before
initialization
let fruit = "kiwi";
```

We get an error, because "regular variables" are not hoisted.

2. Comment out the error, and log fruit after it is declared. Now, it works:

```
console.log(fruit);
```

3. Declare a function, and then call it from above and below the declaration:

```
console.log(sayHello('down there!'));

function sayHello(name) {
    return `Hey, ${name}!`;
}

console.log(sayHello('up there!'));
```

Both function calls work, because the function itself has been hoisted.

function experessions and anonymous functions

- A variable with a function for its value is known as a **function expression**.
- As a variable, a **function expression** doesn't get hoisted.
- A **function expression** must be defined before it can be called or "listened for".
- To make a **function expression**, set a variable equal to an unnamed function.
- A function without a name is known as an **anonymous function**.

4. Write a function expression, and then call it from above and below:

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

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

console.log(welcomeUser('Jane2'));
```

We get an error for the first attempt to call welcomeUser, because a function expression is first and foremost a variable--and variables don't get hoisted.

callbacks: functions as arguments of other functions

A function can take any kind of argument: string, number, boolean--even another function. A function passed to another function as its argument is known as a **callback function**, or simply a **callback**. We have already been working with callbacks without referring to them as such. The addEventListener method takes two arguments: an event, such as "click", and a callback to execute when said event occurs.

5. Get the four buttons, each of which will call a different type of function:

- function declaration: starts with keyword "function" (btn1)
- function expression: variable set equal to anonymous function (btn2)
- inline function: an anonymous function as callback (btn3)
- onclick: a DOM object property set equal to an anonymous function (btn4)

```
const btn1 = document.getElementById('btn-1');
const btn2 = document.getElementById('btn-2');
const btn3 = document.getElementById('btn-3');
const btn4 = document.getElementById('btn-4');
```

6. Get the four span tags, each of which will hold output from one of the four functions:

```
const out1 = document.getElementById('out-1');
const out2 = document.getElementById('out-2');
const out3 = document.getElementById('out-3');
const out4 = document.getElementById('out-4');
```

Button 1: Calling Function Declaration

7. Instruct the first button to call a function when clicked. The function to call on click, that second argument, is a **callback**:

```
btn1.addEventListener('click', onBtn1Click);
```

8. Declare the onBtn1Click function, which outputs a message to the first span tag:

```
function onBtn1Click() {
    out1.textContent = 'Function Declaration says "Hey!"';
}
```

9. Reload the page, and click BTN 1 to get the message.

Button 2: Calling Function Expression

10. Write a function expression by declaring a variable by the name of onBtn2Click and setting it equal to an anonymous function, which outputs a message to the second span tag:

```
const onBtn2Click = function() {
    out2.textContent = 'Function Expression says "Hola!"';
}
```

11. Instruct the second button to call the function when clicked. Notice that we had to do the listener AFTER the expression, since function expressions are variables--and variables cannot be refernced before they are declared.

```
btn2.addEventListener('click', onBtn2Click);
```

12. Reload the page, and click BTN 2 to get the message.

Inline Anonymous Functions

In a function expression, a variable is set equal to a function. Since the variable provides the name, the function value itself is anonymous. An anonymous function can also be a callback, that is, the argument of another function. The addEventListener method takes a callback as its second argument.

We will instruct the third button to run an anonymous function inline, that is, right there inside the addEVentListener method. This is in contrast to our usual practice of having the addEVentListener call an external function by name.

Button 3: Running an Inline Anonymous Function

13. Have the third button listen for a click. When the click occurs, run an **inline anonymous function** right there on the spot:

```
```

```
btn3.addEventListener('click', function() {
    out3.textContent = 'Inline anon function says "Hi!"';
})
```

14. Reload the page, and click BTN 3 to get the message.

onclick property

Any **event** that can be called on a DOM object co-exists as a property of that object. A button can run a function on "click", and therefore has an **onclick** property. If you set the onclick property equal to an anonymous function, when you click the button, the function will run.

Button 4: onclick property = anonymous function

15. Set the fourth button's onclick property equal to an anonymous function:

```
btn4.onclick = function() {
    out4.textContent = 'onclick anon function says "Yo!"';
}
```

16. Reload the page, and click BTN 4 to get the message.