Some Examples

WE'LL COVER THE FOLLOWING

- Checking that a Particular Key Was Pressed
- Doing Something When the Arrow Keys are Pressed
- Detecting Multiple Key Presses

Now that you've seen the horribly boring basics of how to work with Keyboard events, let's look at some examples that clarify (or potentially confuse!) everything you've seen so far.

Checking that a Particular Key Was Pressed

The following example shows how to use the `keyCode` property to check if a particular key was pressed:

```
window.addEventListener("keydown", checkKeyPressed, false);

function checkKeyPressed(e) {
    if (e.keyCode == "65") {
        alert("The 'a' key is pressed.");
    }
}
```

The particular key I check is the **a** key. Internally, this key is mapped to the `keyCode` value of 65. You can find a handy list of all key and character codes at the following link. Please do not memorize every single code from that list. There are far more interesting things to memorize instead.

Some things to note. The `charCode` and `keyCode` values for a particular key are not the same. Also, the `charCode` is only returned if the event that triggered your event handler was `keypress`. In our example, the `keydown` event would not contain anything useful for the `charCode` property.

If you wanted to check the `charCode` and use the `keypress` event, here is what the above example would look like:

```
window.addEventListener("keypress", checkKeyPressed, false);

function checkKeyPressed(e) {
    if (e.charCode == "97") {
        alert("The 'a' key is pressed.");
    }
}
```

The `charCode` for the **a** key is 97. Again, refer to the table of key and character codes I listed earlier for such details.

Doing Something When the Arrow Keys are Pressed

You see this most often in games where pressing the arrow keys does something interesting. The following snippet of code shows how that is done:

```
window.addEventListener("keydown", moveSomething, false);

function moveSomething(e) {
    switch(e.keyCode) {
        case 37:
            // left key pressed
            break;
        case 38:
            // up key pressed
            break;
        case 39:
            // right key pressed
            break;
        case 40:
            // down key pressed
            break;
    }
}
```

Again, this should be pretty straightforward as well. The only potentially weird thing is the `switch` statement, and you can learn more about them in this tutorial!

Detecting Multiple Key Presses

Now, this is going to be epic! An interesting case revolves around detecting when you need to react to multiple key presses. Below is an example of how to do that:

```
window.addEventListener("keydown", keysPressed, false);
window.addEventListener("keyup", keysReleased, false);

var keys = [];

function keysPressed(e) {
    // store an entry for every key pressed
    keys[e.keyCode] = true;

    // Ctrl + Shift + 5
    if (keys[17] && keys[16] && keys[53]) {
        // do something
    }

    // Ctrl + f
    if (keys[17] && keys[70]) {
        // do something

        // prevent default browser behavior
        e.preventDefault();
    }
}

function keysReleased(e) {
    // mark keys that were released
    keys[e.keyCode] = false;
}
```

Going into great detail about this will require another tutorial by itself, but let's just look at how this works.

First, we have a `keys` array that stores every single key that you press:

```
var keys = [];
```

As keys get pressed, the `keysPressed` event handler gets called:

```
function keysPressed(e) {
    // store an entry for every key pressed
    keys[e.keyCode] = true;
}
```

When a key gets released, the `keysReleased` event handler gets called:

```
function keysReleased(e) {
```

```
    // mark keys that were released
    keys[e.keyCode] = false;
}
```

Notice how these two event handlers work with each other. As keys get pressed, an entry gets created for them in the `keys` array with a value of **true**. When keys get released, those same keys are marked with a value of **false**. The existence of the keys you press in the array is superficial. It is the values they store that is actually important.

As long as nothing interrupts your event handlers from getting called properly such as an alert window, you will get a one-to-one mapping between keys pressed and keys released as viewed through the lens of the `keys` array. With all of this said, the checks for seeing which combination of keys have been pressed is handled in the `keysPressed` event handler. Lines 5-16 show how this works:

```
function keysPressed(e) {
    // store an entry for every key pressed
    keys[e.keyCode] = true;

    // Ctrl + Shift + 5
    if (keys[17] && keys[16] && keys[53]) {
        // do something
    }

    // Ctrl + f
    if (keys[17] && keys[70]) {
        // do something

        // prevent default browser behavior
        e.preventDefault();
    }
}
```

There are two things you need to keep in mind. The order of your checks matter. Ensure the checks are arranged in decreasing order of the number of keys that are pressed. Second, some key combinations result in your browser doing something. To avoid your browser from doing its own thing, use the `preventDefault` method like I show when checking to see if Ctrl + F is being used:

```
function keysPressed(e) {
    // store an entry for every key pressed
    keys[e.keyCode] = true;
```

```
    // Ctrl + Shift + 5
    if (keys[17] && keys[16] && keys[53]) {
        // do something

    }

    // Ctrl + f
    if (keys[17] && keys[70]) {
        // do something

        // prevent default browser behavior
        e.preventDefault();
    }
}
```

The `preventDefault` prevents your browser from reacting to it by showing the Find dialog for Ctrl + f. You put all of this together, and you have a basic blueprint for how to check for multiple key presses easily.