Sound Check Experiment

This is a preliminary test for making sound with javascript. Right now, the code will be tangled together in an attempt to understand how everything works. The actual program will have much more separation of parts.

Global Constants

The Lists will be constant throughout the whole program. The program will randomly select an item from one of these lists depending on the situation, and use that value to affect the sound in some way.

```
const LIST_OF_WAVEFORM_TYPES = [
    'sine', 'square', 'sawtooth', 'triangle'
];
```

Global Functions

During the creation of random sounds, randomly generating numbers is a common occurence. This function generates a random integer between two integers.

More precisely, let $f(m, M)$ be a function $f : (\mathbb{Z}, \mathbb{Z}) \to \mathbb{Z}$ such that $f(m, M) = x$ and $m \leq x \leq M$

```
function getRandomInt(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max) + 1;
    return Math.floor(Math.random() * (max - min)) + min;
}
```

Since the most common target is a random value from one of the global Constant Arrays, a useful function will return one of those randomly selected values. This can be achieved by picking a random KEY in the array, with the minimum being 0, and the maximum being the length of the array.

```
function getRandomValueFromArray(name) {
    return name[ getRandomInt(0, name.length - 1) ];
}
```

Audio Context

Quote from Mozilla Developer - Audio Context

> The AudioContext interface represents an audio-processing graph built from audio modules linked together, each represented by an AudioNode. An audio context controls both the creation of the nodes it contains and the execution of the audio processing, or decoding. You need to create an AudioContext before you do anything else, as everything happens inside a context.

```
var audioContext = new (window.AudioContext || window.webkitAudioContext)();
```

Simple Oscillator

For this example, an oscillator node will be created. Given that we have a list of waveforms, we will choose one of them at random upon creation of the oscillator.

```
var buildRandomOscillator = function () {
    var osc;
    osc = audioContext.createOscillator();
    RandomizeOscillator(osc);
    osc.start(audioContext.currentTime);
    console.log('New Oscillator Created!\n', osc);
    return osc;
};
```

RandomizeOscillator will take an existing oscillator and give it random values for the type and frequency.

```
var RandomizeOscillator = function (osc) {
    osc.type = getRandomValueFromArray(LIST_OF_WAVEFORM_TYPES);
    osc.frequency.value = getRandomInt(400,100);
    return osc;
};
```

Button Events

Start and Stop the Sound

For now, we want an oscillator to be defined globally, so that we can turn it on or off. Before any buttons are pressed, there is no sound. We just have an empty variable where the oscillator will go when it is turned on.

```
var osc;
var is_sound_on = false;
osc = buildRandomOscillator();
```

Starting the sound will *Connect* the oscillator to the speakers, effectively turning it on for the listener.

```
var startSound = function() {
    osc.connect(audioContext.destination);
    is_sound_on = true;
};
```

Ending the sound will *Disconnect* the oscillator from the speakers, which will effectively turn the sound off.

```
var endSound = function() {
    osc.disconnect(audioContext.destination);
    is_sound_on = false;
};
```

Toggle sound simply decides if the audio should be turned on or off based on its current status.

```
var toggleSound = function () {
    if  (!is_sound_on)   {return startSound();}
    if  (is_sound_on)    {return endSound();}
};
```

Changing the Interface

updateInterface() will call of the text-altering functions that affect what is displayed on the screen. Globalized variables are converted into strings, and then into HTML.

Note: Functions outside of the interface call upon this one. It connects the actual sound events to the interface. Everything else, and how the interface works, is defined further in this section.

```
var updateInterface = function () {
    changeSoundStatus(is_sound_on);
    changeTextWaveform(is_sound_on, osc.type);
    changeTextFrequency(is_sound_on, osc.frequency.value);
};
```

Map TRUE and FALSE boolean values into strings

```
var boolHighL = function (status) {
    if (status) {return 'Highlight';}
    else        {return 'NoHighlight';}
};
```

These functions interact with the interface (which is the browser).

```
var changeSoundStatus = function (status) {
    var span = document.getElementById('sound_status');
    span.innerHTML = status.toString();
    span.className = status.toString();
};
var changeTextWaveform = function (status, waveform) {
    document.getElementById('waveform_type').innerHTML = waveform;
    document.getElementById('waveform_type').className = boolHighL(status);
};
var changeTextFrequency = function (status, frequency) {
    document.getElementById('span_frequency').innerHTML = frequency.toString();
    document.getElementById('span_frequency').className = boolHighL(status);
};
var humanHearingCheck = function(frequency) {
    return;
};
```