Class 6

- libuv multithreaded library adds asyncronous taks to the task queue. These tasks will finish in an unordered sequence which will vary from time to time however, non syncronous tasks will always finish first.

```
for (let i = 0; i < 10000000000; i++) {
  // blank
}
console.log("finished!");

/*
this for loop is a blocking function. When you execute this code, the rest
of your code will freeze up from entering the v8 stack until the for loop
is finished.
*/

setTimeout(function () {
  console.log("finished");
}, 1000);

/*
this is a non blocking code. Instead of freezing up the client side, this
task will get sent to te libuv queue while other interactions with the
front end/ back end will continue to work.
*/
```

> Asyncronously

```
/*
This will print out of order because each task is completing
asyncronously.
*/
const fs = require("fs");

const { writeFile } = fs;

/*
if we use var instead of let in the scope of the for loop, its going to
print 99 for every output.txt file. This is because the var keyword only
preserves one slot in memory for the variable i
 */
for (let i = 0; i < 100; i++) {
  writeFile(
    `data1/${i < 10 ? "0" : ""}${i}-output.txt`,
    "Data-1",
    (data, err) => {
      if (err) {
        console.log("error");
```

```
      } else {
        console.log(
          `${i < 10 ? "0" : ""}${i}-output.txt has finished writing..`
        );
      }
    }
  );
}
```

Syncronously

```
/*
this will print in order because the function calls are always going to
finish before another function gets called. This causes blocking code that
can effect other code from entering the v8 stack.
*/
let count = 0;
function write() {
  writeFile(
    `data2/${count < 10 ? "0" : ""}${count}-output.txt`,
    "Data-2",
    callback
  );
}

function callback(data, err) {
  console.log(
    `${count < 10 ? "0" : ""}${count}-output.txt has finished writing..`
  );
  count++;
  if (count == 100) {
    console.log("All files finished printing..");
    return;
  }
  write();
}

write();
```