JS Communications
Node.js

Non-Blocking in Javascript

- Anything that could block is handled asynchronously
- Generally this means creating an anonymous function containing the things you want to do async
- The anon function gets evaluated at load, but the function is not executed until the async event triggers

```
<html>
    <script language="javascript">
        let runTimer=false;
        let pollCount=0;
        function stoptime() { runTimer = false; }
        function dotime() {
            if (runTimer) { return; }
            runTimer=true;
            function innerTimer() {
                fetch("http://165.106.10.170:30009/short")
                .then(function(response) {
                    pollCount++;
                    response.text()
                    .then(function(text) {
                        if (!runTimer) { return; }
                        console.log(`Poll Count ${pollCount}`)
                        console.log(text);
                        setTimeout( function() { innerTimer(); }, 2000);
                    });
                });
            }
            setTimeout(function() { innerTimer() }, 2000);
        }
    </script>
    <body>
        <button type="submit" onclick="dotime();">Start Polling</button>
        <br><button type="submit" onclick="stoptime();">Stop Polling</button>

    </body>
</html>
```

Anonymous function

Anonymous function

Fetch. Javascript to get information from a website

Anonymous function

Anonymous function

Most languages have blocking constructs

- Most languages have some form of parallel execution
  - "Thread"
  - So even if one thread is blocked thing can keep happening
    - For example, a simple hello world server in Go with a 15 second sleep block.
    - Problems with using threads to solve blocking?

- Android / iOS both disallow blocking in the main IO thread

```go
package main

import (
    "fmt"
    "net/http"
    "time"
)


func hw(w http.ResponseWriter, req *http.Request) {
    fmt.Println("Enter hello")
    time.Sleep(15*time.Second)
    fmt.Fprintf(w, "hello world")
    fmt.Println("Exit hello")
}


func main() {
    http.HandleFunc("/hello", hw)
    http.ListenAndServe(":30030", nil)
}
```

Writing SQL functions

- Often SQL functions are simple things just to reduce complexity in a query
- In the hurricane database, write a function that returns the max wind speed observed for a given hurricane
  - In use: select hid, name, maxx_speed(hid) from hurricane;
    - *This is a bad use of procedures (it is really slow!!!) Why?*
- In the hurricane database, write a function that gives the power output of a wind turbine for given wind speed.
  - The function is P = 0.000133 * 0.4 * Area * V*V*V
    - Area is the area spanned by rotor of a given length (pi*R*R)
  - In use: select hid, date, time, max_sustained, powerr(max_sustained, 200) from observation;
    - where 200 is the length (in feet) of the wind turbine blade.