Node.js

what is Node?

- a JavaScript runtime environment for server-side and networking applications
- open source platform built on Google Chrome's V8 engine
- event-driven, non-blocking model for building data-intensive real-time applications
- a way to unify the web stack - programmers can use the same language to build client and server

what Node isn't

- a platform for doing computationally expensive tasks
- a framework (doesn't replace Rails or Django) Node is a platform, but a minimal one, providing access to mostly low-level operations
- multi-threaded - think of Node as a single-threaded server

who is using Node?

- Yahoo
- PayPal
- Microsoft
- Uber
- Dow Jones
- The New York Times
- EBay
- Linkedin
- AirBnB
- Starbucks

what can you build with Node?

- WebSocket server (like a chat server)
- File upload client
- Ad server
- Any real-time data application
- network servers

Node has a REPL available:

```
node
>
```

blocking vs. non-blocking: an example with file I/O

1. open a file
2. read its contents and print
3. do something else

blocking vs. non-blocking: simulating long-running file I/O

1. run a function that waits 5 seconds then opens a file
2. read its contents and print
3. do something else

examples of typical blocking operations:

- calls out to web services
- reads and writes to a database
- file I/O

building a simple http server in Node - server.js

```
var http = require('http');

http.createServer(function (request, response) {

    response.writeHead(200, { 'Content-Type': 'text/plain' });
    response.write('Hello, class!');
    response.end('Goodbye, class!');
}).listen(3000, function () {
    console.log('listening on port 3000');
});
```

equivalent to:

```
var http = require('http');


function handleRequest(request, response) {

    response.writeHead(200, { 'Content-Type': 'text/plain' });
    response.write('Hello, class!');
    response.end('Goodbye, class!');
}

http.createServer(handleRequest).listen(3000, function () {
    console.log('listening on port 3000');
});
```

As Node is executing the code, it will register events. In this case, we're listening for and responding to the 'request' event. The server we've created emits other events as well, such as 'connection', or 'close'.

After executing the script, Node will go into an event loop, constantly checking if an event has been emitted, and if so, executing any callbacks registered with that event.

explicitly handling events:

```
var http = require('http');


var server = http.createServer();

server.on('request', function (response) {

    response.writeHead(200, { 'Content-Type': 'text/plain' });
    response.write('Hello, class!');
    response.end('Goodbye, class!');
});

server.listen(3000, function () {
    console.log('listening on port 3000');
});
```

Events in Node

Events in Node are very similar to how events are handled in the browser. Objects that emit events in Node typically inherit from EventEmitter.

demonstrating EventEmitter:

```
var EventEmitter = require('events').EventEmitter;
var emitter = new EventEmitter();

emitter.on('helloEvent', function (message) {
    console.log('helloEvent emitted!');
    console.log('helloEvent message: ', message);
});

emitter.emit('helloEvent', 'Hello, class!');
```

similar to DOM events, multiple listeners can be established for a given event:

```
var EventEmitter = require('events').EventEmitter;
var emitter = new EventEmitter();

emitter.on('helloEvent', function (message) {
    console.log('helloEvent emitted!');
    console.log('helloEvent message: ', message);
});

emitter.on('helloEvent', function (message) {
    console.log('the helloEvent message was ' + message.length +
            ' characters long');
});

emitter.emit('helloEvent', 'Hello, class!');
```