Date
Page

~~* Creating a Module:~~

→ Node is a C++ prog. it includes chrome's V8 Javascript engine (it parses & executes Js code).
So, when we pass app.js file to node, it's going to give it to V8 for execution.

→ ~~Co~~ e.g:
console.log(module); // this modules seems to be public/global but it's private

O/P: Module{ → object Module, it's a json obj. with key value pairs, so we've id (unique identifier), exports, parents, filename (complete path to that file), loaded (boolean that determines if this module is loaded or not), children & paths

```
O/P: Module{
  id: '.',
  exports: {},
  parent: null,
  filename: '              ',
  loaded: false,
  children: [],
  paths:
  [
     ] }
```

* Creating Modules:

→ ~~The exports is empty~~ The exports property is set to an empty obj., anything that we add to this object will be exported ~~to~~ from this module & will be available outside of this module (public).

eg:

```
var url = '~~ht~~https://mylogger./log';
```

Implementation detail.

```
fn. log(message){
    // send an HTTP request
    console.log(message);
}
```

```
module.exports.log = log;
```

Adding method called log to this exports obj. & setting it to this log fn. we've defined here. In other words the obj. that we're exporting here has a single method called log.

```
module.exports.url = url;
```

We can ~~define only~~ export any thing this way.

```
module.exports.endPointurl = url;
```

we can change the name that's exported to the outside like here we did, we change url to endPointurl.

we may call this variable URL but when we export it we may call it endpointURL.

So, in real world applications, every module might have several variables & functions. We only want to export a subset of these members to the outside, coz we want to keep this module easy to use.

So, in our logger module (logger.js file) this URL is implementation detail, other modules don't need to know anything about this, they only need to call log fn.
So, we export these make it public ~~&~~ but keep the URL private