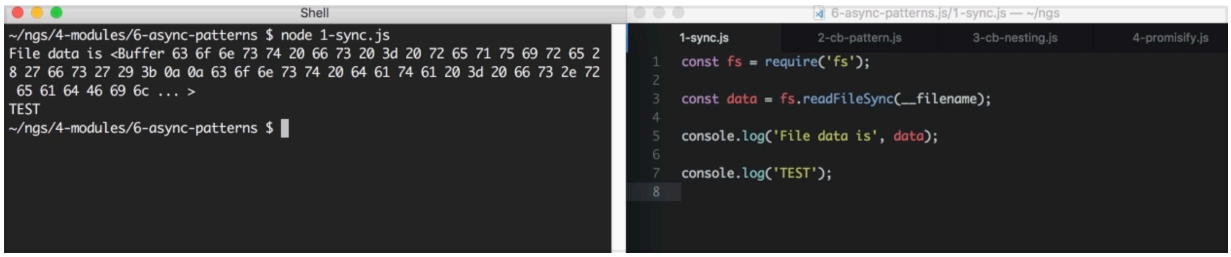
Node

```
setTimeout(
  () => {
    console.log('Hello after 4 seconds');
  },
  4 * 1000
);
```

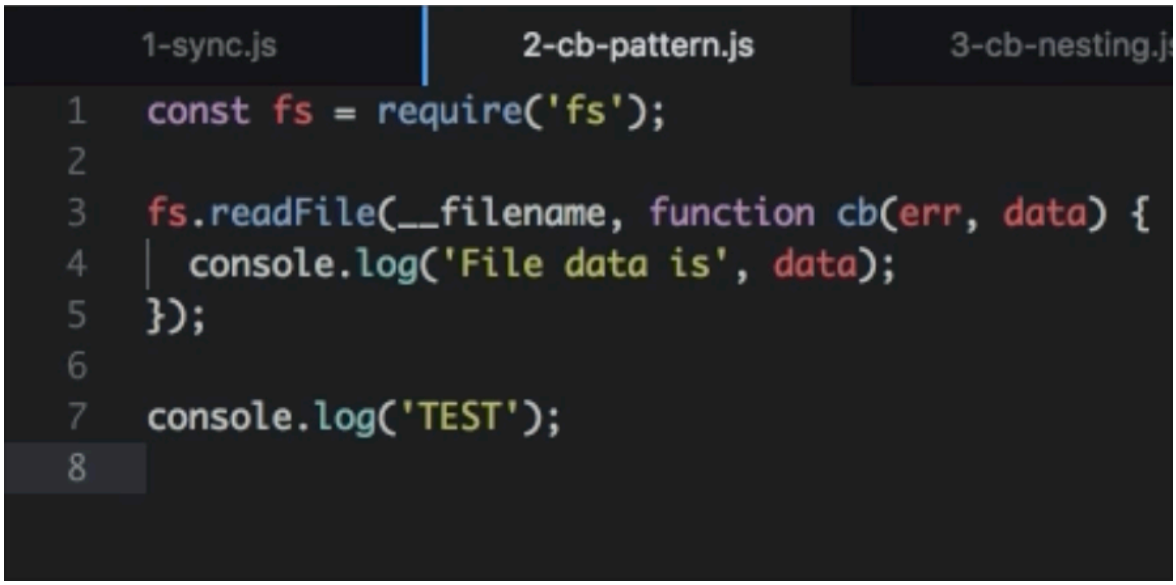
```
const func = () => {
  console.log('Hello after 4 seconds');
};

setTimeout(func, 4 * 1000);


// For: func(arg1, arg2, arg3, ...)
// We can use: setTimeout(func, delay, arg1, arg2, arg3, ...)
```

```
const rocks = who => {
  console.log(who + ' rocks');
};

setTimeout(rocks, 2 * 1000, 'Pluralsight');
```

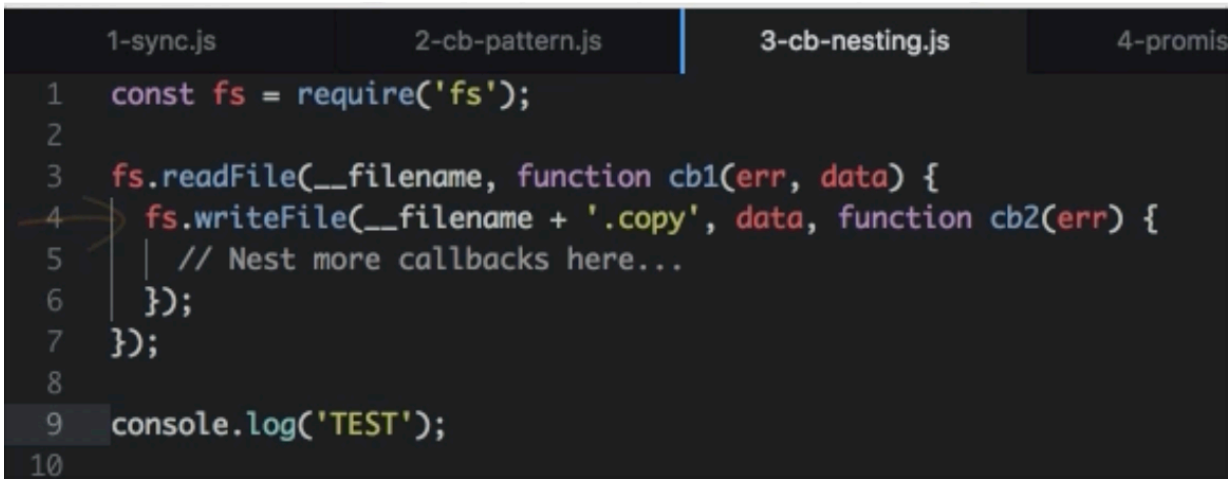
Sync execution —



Async - call back



Nested call back



Promise

5-fs-promises.js

```
1  const { readFile } = require('fs').promises;
2
3  async function main() {
4    const data = await readFile(__filename);
5    console.log('File data is', data);
6  }
7
8  main();
9
10 console.log('TEST');
11
```

6-promise-nesting.js

```
1  const fs = require('fs').promises;
2
3  async function main() {
4    const data = await fs.readFile(__filename);
5    await fs.writeFile(__filename + '.copy', data);
6    // More awaits here...
7  }
8
9  main();
10 console.log('TEST');
11
```

Types of Patterns

Creational
1) Constructor
2) Module
3) Factory
4) Singleton

Structural
1) Decorator
2) Façade
3) Flyweight

Behavioral
1) Command
2) Mediator
3) Observer

Creating Objects

```
var obj = {};
var nextObj = Object.create(Object.prototype);
var lastObj = new Object();
```

Creating a New Object

```
var obj = {};
obj.param = 'new value';
console.log(obj.param); // new value
```

Assigning Keys and Values

Dot notation

```
var obj = {};
obj['param'] = 'new value';
console.log(obj['param']); // new value
```

Assigning Keys and Values

Square bracket notation

```
var task = Object.create(Object.prototype);

task.title = 'My task';
task.description = 'My Description';
task.toString = function(){
    return this.title + ' ' + this.description;
}

console.log(task.toString());
```

```
var task = {
title: 'My Title',
description: 'My Description'
};
task.toString = function(){
    return this.title + ' ' + this.description;
}

console.log(task.toString());
```

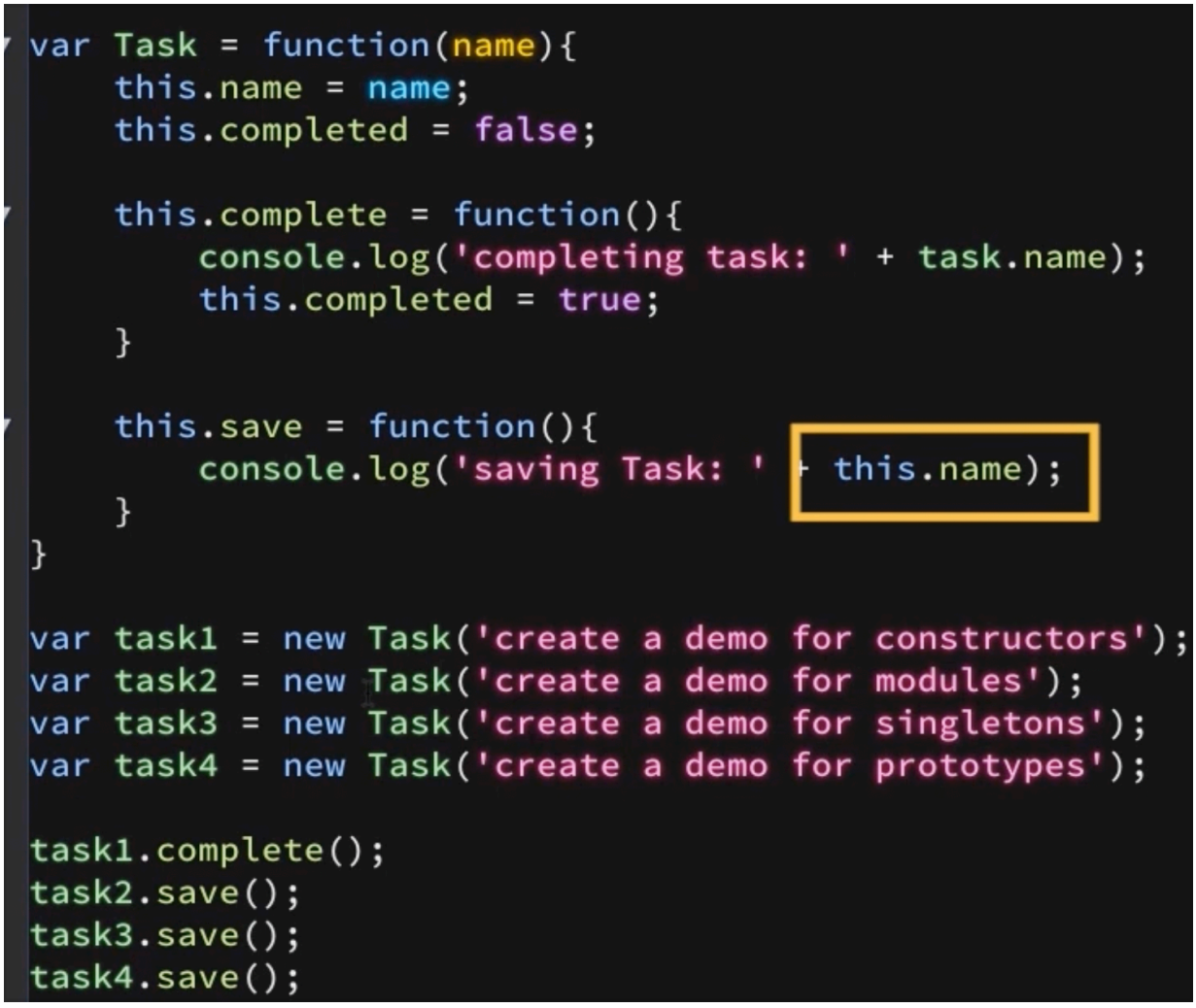
Defining properties

```
var task = {
    title: 'My Title',
    description: 'My Description'
};

Object.defineProperty(task, 'toString', {
    value: function () {
        return this.title + ' ' + this.description;
    },
    writable: true,
    enumerable: true,
    configurable: true
});



console.log(task.toString());
```

```
function ObjectName(param1, param2){
    this.param1 = param1;
    this.param2 = param2;
    this.toString = function () {
        return this.param1 + ' ' + this.param2;
    }
}
```

Constructor Pattern

Create objects from functions.

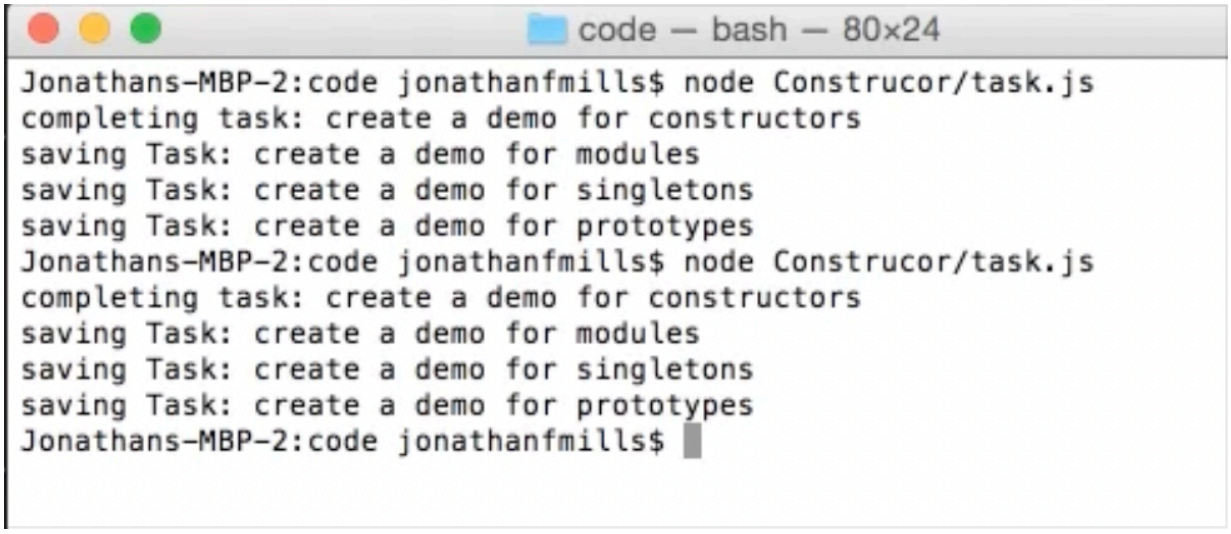


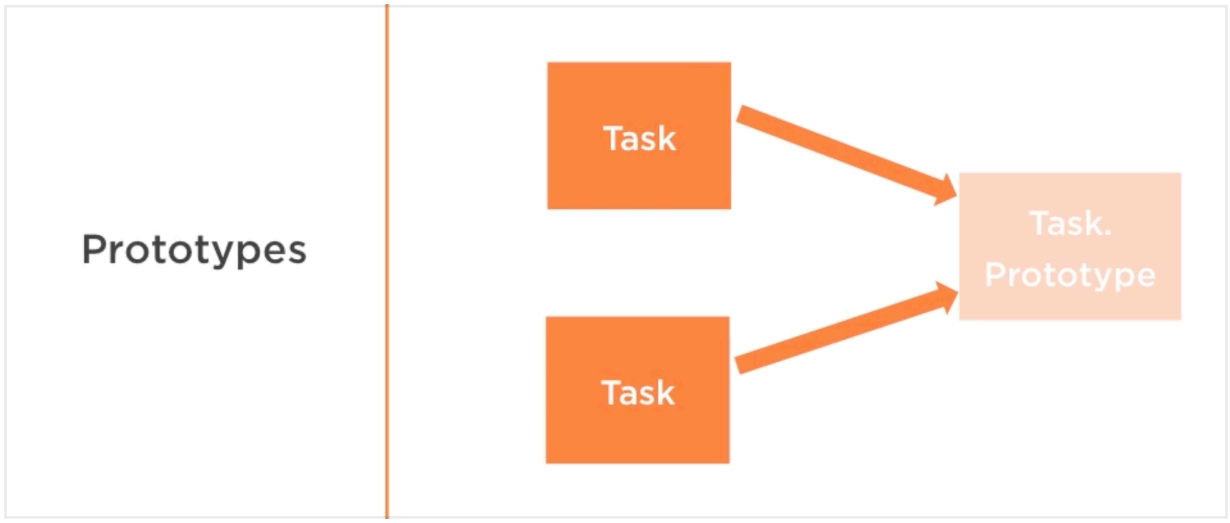
```
var Task = function(name){
    this.name = name;
    this.completed = false;

    this.complete = function(){
        console.log('completing task: ' + task.name);
        this.completed = true;
    }

    this.save = function(){
        console.log('saving Task: ' + this.name);
    }
}

var task1 = new Task('create a demo for constructors');
var task2 = new Task('create a demo for modules');
var task3 = new Task('create a demo for singletons');
var task4 = new Task('create a demo for prototypes');

task1.complete();
task2.save();
task3.save();
task4.save();
```

Prototype pattern

```
var Task = function (name) {
    this.name = name;
    this.completed = false;
}

Task.prototype.complete = function () {
    console.log('completing task: ' + this.name);
    this.completed = true;
};

Task.prototype.save = function () {
    console.log('saving Task: ' + this.name);
};
var task1 = new Task('create a demo for constructors');
var task2 = new Task('create a demo for modules');
var task3 = new Task('create a demo for singletons');
var task4 = new Task('create a demo for prototypes');

task1.complete();
task2.save();
task3.save();
task4.save();
```

Output same for con





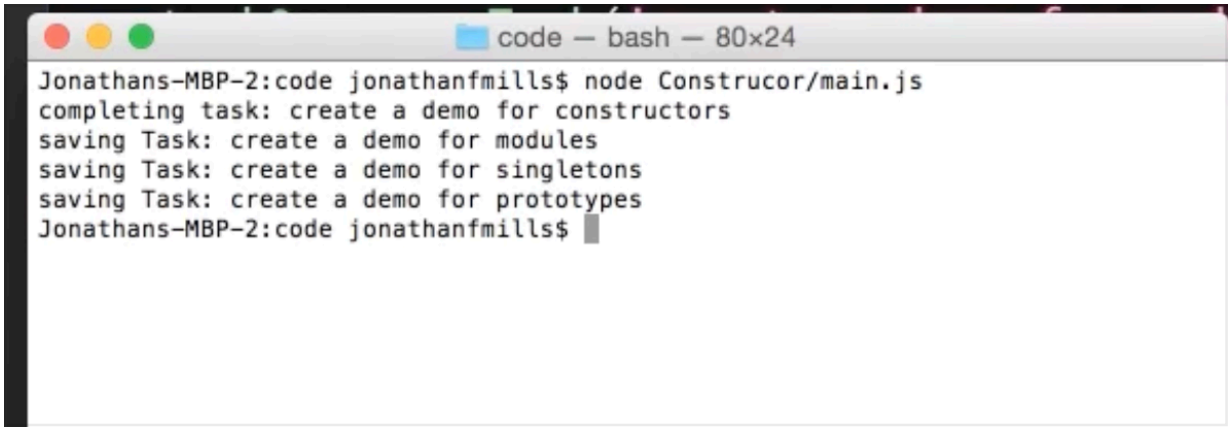
Constructor Node :

task.js main.js

```
1  var Task = function (name) {
2      this.name = name;
3      this.completed = false;
4  }
5
6  Task.prototype.complete = function () {
7      console.log('completing task: ' + this.name);
8      this.completed = true;
9  };
10
11 Task.prototype.save = function () {
12     console.log('saving Task: ' + this.name);
13 };
14
15 module.exports = Task;
```

• { } main.js

```
var Task = require('./task');

var task1 = new Task('create a demo for constructors');
var task2 = new Task('create a demo for modules');
var task3 = new Task('create a demo for singletons');
var task4 = new Task('create a demo for prototypes');

task1.complete();
task2.save();
task3.save();
task4.save();
```



Class:

{} class.js

```
3  class Task {
4      constructor(name) {
5          this.name = name;
6          this.completed = false;
7      };
8      complete() {
9          console.log('completing task: ' + this.name);
10         this.completed = true;
11     };
12
13     save() {
14         console.log('saving Task: ' + this.name);
15     };
16 }
17
18 var task1 = new Task('create a demo for constructors');
19 var task2 = new Task('create a demo for modules');
20 var task3 = new Task('create a demo for singletons');
21 var task4 = new Task('create a demo for prototypes');
22
23 task1.complete();
24 task2.save();
25 task3.save();
26 task4.save();
```

Moduler Pattern:

{} taskRepository.js {} main.js {} tas...

```
1  var repo = function () {
2
3      return {
4          get: function(id) {
5              console.log('Getting task ' + id);
6              return {
7                  name: 'new task from db'
8              }
9          },
10         save: function(task){
11             console.log('Saving ' + task.name + ' to the db');
12         }
13     }
14
15
16 }
17
18 module.exports = repo();
```

```
var Repo = require('./taskRepository');

var Task = function (data) {
    this.name = data.name;
    this.completed = false;
}

Task.prototype.complete = function () {
    console.log('completing task: ' + this.name);
    this.completed = true;
};

Task.prototype.save = function () {
    console.log('saving Task: ' + this.name);
    Repo.save(this);
};

module.exports = Task;
```

taskRepository.js | main.js | task.js

```
1  var Task = require('./task');
2  var Repo = require('./taskRepository');
3
4  var task1 = new Task(Repo.get(1));
5
6  var task2 = new Task('create a demo for modules');
7  var task3 = new Task('create a demo for singletons');
8  var task4 = new Task('create a demo for prototypes');
9
10 task1.complete();
11 task2.save();
12 task3.save();
13 task4.save();
```

```
saving Task: undefined
Jonathans-MBP-2:code jonathanfmills$ node Module/main.js
Getting task 1
completing task: new task from db
saving Task: undefined
Saving undefined to the db
saving Task: undefined
Saving undefined to the db
saving Task: undefined
Saving undefined to the db
Jonathans-MBP-2:code jonathanfmills$
```

Factory Pattern

A pattern used to simplify object creation.

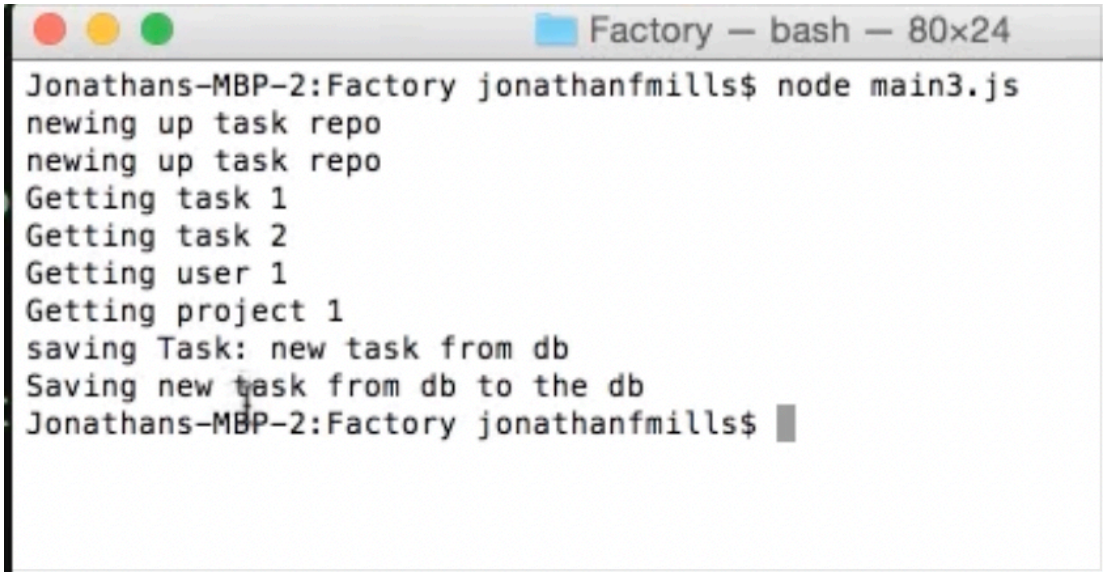
Factory/repoFactory2.js (code) — Brackets

{} main.js | {} repoFactorywCache.js | {} main2.js | {} main3.js | {} repoFactory2.js

```
1  var repoFactory = function(){
2      var repos = this;
3      var repoList = [{name:'task', source:'./taskRepository'},
4                      {name:'user', source:'./userRepository'},
5                      {name:'project', source:'./projectRepository'}];
6
7      repoList.forEach(function(repo){
8          repos[repo.name] = require(repo.source)()
9      });
10 };
11
12
13 module.exports = new repoFactory;
```

{} repoFactorywCache.js | {} main2.js | {} main3.js | {} repoFactory2.js

```
var Task = require('./task');
var repoFactory = require('./repoFactory2');

var task1 = new Task(repoFactory.task.get(1));
var task2 = new Task(repoFactory.task.get(2));

var user = repoFactory.user.get(1);
var project = repoFactory.project.get(1);

task1.user = user;
task1.project = project;

//console.log(task1);
task1.save();
```

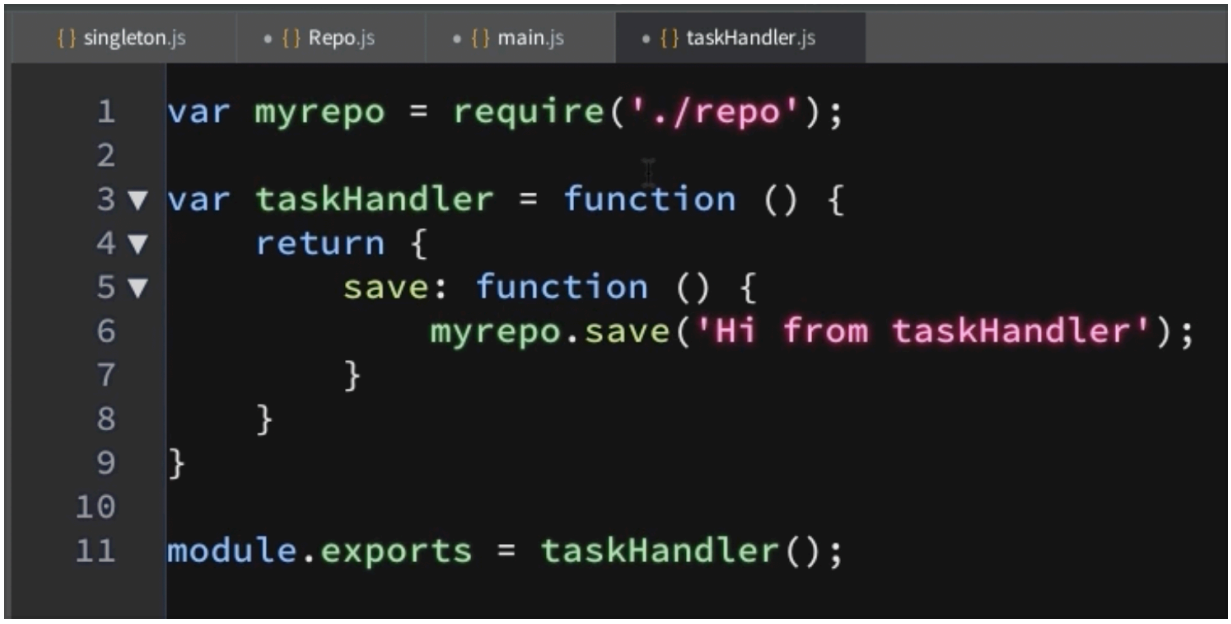


Singleton :

{} singleton.js • {} Repo.js • {} main.js {} taskHandler.js

```
1  var taskHandler = require('./taskHandler');
2  var myrepo = require('./repo');
3
4  myrepo.save('fromMain')
5  myrepo.save('fromMain')
6  myrepo.save('fromMain')
7  taskHandler.save();
8  taskHandler.save();
9  taskHandler.save();
10 taskHandler.save();
11
```

```
var repo = function () {

    var called = 0;

    var save = function (task) {
        called++;
        console.log('Saving ' + task +
                    ' Called ' + called + ' times');
    }
    console.log('newing up task repo');
    return {
        save: save
    }
}
module.exports = repo();
```



#pluralsight