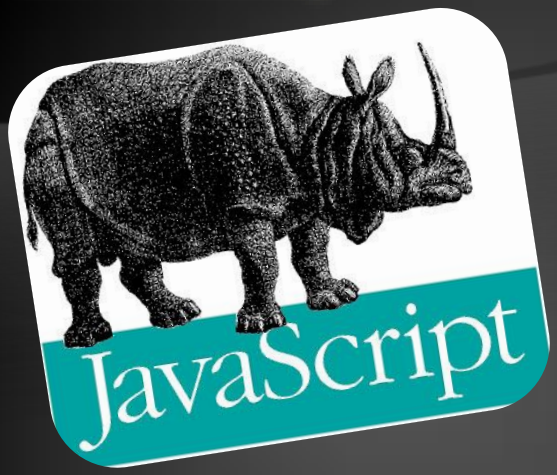
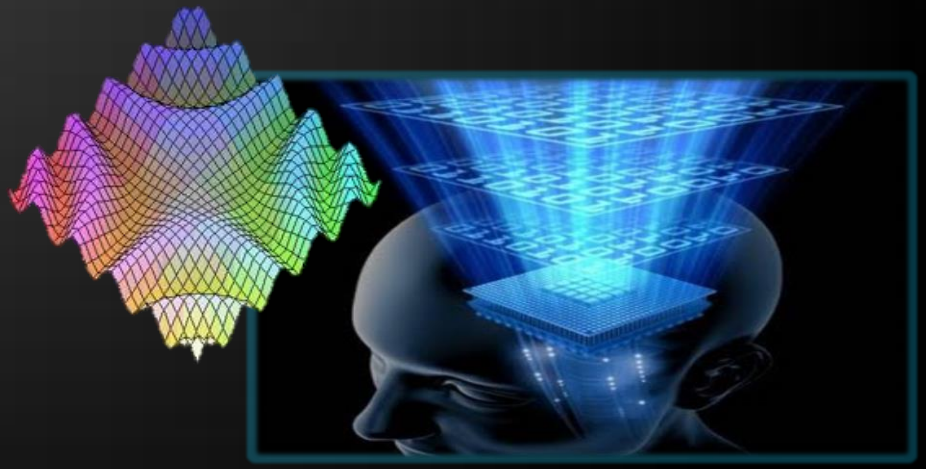
Telerik Academy



JavaScript Patterns

Modules and stuff



JavaScript OOP

Telerik Software Academy

http://academy.telerik.com

Telerik Academy

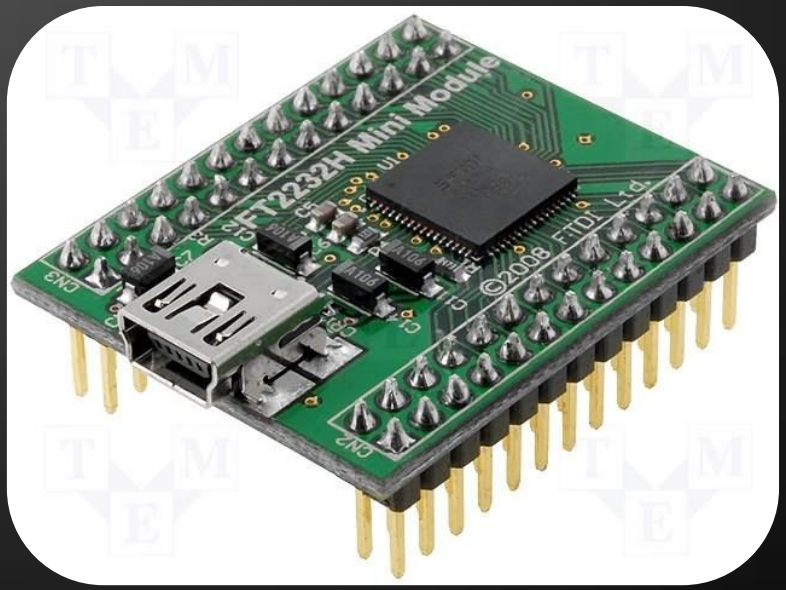
Table of Contents

1. Public/Private fields in JavaScript
2. Module pattern
3. Revealing module pattern
4. Singleton pattern

Telerik Academy

The Module Pattern

Hide members



Pros and Cons

- Pros:
  - "Modularize" code into re-useable objects
  - Variables/functions not in global namespace
  - Expose only public members
- Cons:
  - Not easy to extend
  - Some complain about debugging

Module Pattern: Structure

- Structure:

```
var module = (function() {
        //private variables
        //private functions

        return {
                //public members
                someFunc: function() {…},
                anotherFunc: function() {…}
        };
}());
```

Module Pattern: Example

- Example:

```
var controls = (function () {
  function formatResult(name, value) {
    return name + ' says the result is ' + value;
  }
  var calculator = {
    init: function (name) { /* init code */ },
    add: function (x) { /* code to add */ },
    subtract: function (x) { /* code to subtract */ },
    showResult: function () { /* code to show result */ }
  };
  return {
    getCalculator: function (name) {
      return Object.create(calculator)
        .init(name);
    }
  };
} ());
controls.getCalculator('First')
  .add(7) .showResult() .subtract(2) .showResult();
```

Module Pattern: Example

- Example:

```
var controls = (function () {
  function formatResult(name, value) {
    return name + ' says the result is ' + value;
  }
  var calculator = {
    init: function (name) { /* init co
    add: function (x) { /* code to add
    subtract: function (x) { /* code to subtract */ },
    showResult: function () { /* code to show result */ }
  };
  return {
    getCalculator: function (name) {
      return Object.create(calculator)
        .init(name);
    }
  };
} ());
controls.getCalculator('First')
  .add(7) .showResult() .subtract(2) .showResult();
```

The visible members create closures with them

Module Pattern: Example

- Example:

```
var controls = (function () {
  function formatResult(name, value) {
    return name + ' says the result is ' + value;
  }
  var calculator = {
    init: function (name) { /* init code */ },
    add: function (x) { /* code to add */ },
    subtract: function (x) { /* code to subtract */ },
    showResult: function () { /* code to show result */ }
  };
  return {
    getCalculator: function (name) {
      return Object.create(calculator)
        .init(name);
    }
  };
} ());
controls.getCalculator('First')
  .add(7) .showResult() .subtract(2) .showResult();
```

Visiable method

Module Pattern: Summary

- Module pattern provides encapsulation of variables and functions
- Provides a way to add visibility (public versus private) to members
- Each object instance creates new copies of functions in memory

Telerik Academy

Module Pattern

Live Demo

Telerik Academy

The Revealing Module Pattern

Reveal the most interesting members

Revealing Module Pattern: Pros and Cons

- Pros:
  - "Modularize" code into re-useable objects
  - Variables/functions taken out of global namespace
  - Expose only visible members
  - "Cleaner" way to expose members
  - Easy to change members privacy
- Cons:
  - Not easy to extend
  - Some complain about debugging
  - Hard to mock hidden objects for testing

Telerik Academy

Revealing Module Pattern: Structure

- Structure:

```
var module = (function() {
        //hidden variables
        //hidden functions

        return {
                //visible members
                someFunc: referenceToFunction
                anotherFunc: referenceToOtherFunction
        };
}());
```

Revealing Module Pattern: Example

- Example:

```
var controls = (function () {
  function formatResult(name, value) {
    return name + ' says the result is ' + value;
  }
  var calculator = {
    init: function (name) { /* init code */ },
    add: function (x) { /* code to add */ },
    subtract: function (x) { /* code to subtract */ },
    showResult: function () { /* code to show result */ }
  };
  function getCalculator(name){
    return Object.create(calculator).init(name);
  }
  return { getCalculator: getCalculator };
} ());
controls.getCalculator('First')
  .add(7) .showResult() .subtract(2) .showResult();
```

Revealing Module Pattern: Example

- Example:

```
var controls = (function () {
  function formatResult(name, value) {
    return name + ' says the result is ' + value;
  }
  var calculator = {
    init: function (name) { /* init code */ },
    add: function (x) { /* code to add */ },
    subtract: function (x) { /* code to subtract */ },
    showResult: function () { /* code to show result */ }
  };
  function getCalculator(name){
    return Object.create(calculator).init(name);
  }
  return { getCalculator: getCalcu
} ());
controls.getCalculator('First')
  .add(7) .showResult() .subtract(2) .showResult();
```

Create the function hidden

Revealing Module Pattern: Example

- Example:

```
var controls = (function () {
  function formatResult(name, value) {
    return name + ' says the result is ' + value;
  }
  var calculator = {
    init: function (name) { /* init code */ },
    add: function (x) { /* code to add */ },
    subtract: function (x) { /* code to subtract */ },
    showResult: function () { /* code to show result */ }
  };
  function getCalculator(name)
    return Object.create(calcu
  }
  return { getCalculator: getCalculator };
} ());
controls.getCalculator('First')
  .add(7) .showResult() .subtract(2) .showResult();
```

Expose (reveal) only references to hidden member

Revealing Module Pattern: Summary

- Module pattern provides encapsulation of variables and functions
- Provides a way to add visibility (public versus private) to members
- Extending objects can be difficult since no prototyping is used

Telerik Academy

```
var salary = function () {


}();
```

Revealing Module Pattern

Live Demo

Telerik Academy

Singleton Pattern

One object to rule them all!

Singleton Pattern: Structure

- Singleton pattern introduces a single instance each time an instance is requested
  - Harder in other languages, in JavaScript every IIFEs forms a singleton
- Structure:

```
var module = function() {
  var instance = { /* code for instance */};
  instance = Object.preventExtensions(instance);
  return {
    get: function(){
      return instance;
    }
  };
}();
```

Singleton Pattern: Example

- Example:

```
var calculator = (function () {
  var calculator = {
    result: 0,
    add: function (x) { /* code for add */ },
    subtract: function (x) { /* code for subtract */ },
    showResult: function () { /* show result */ }
  };
  calculator = Object.preventExtensions(calculator);
  return { get: function () { return calculator; } };
} ());

calculator.get() .add(7) .subtract(17) .showResult();
//result is -10
calculator.get() .add(111) .showResult();
//result is 101 (continues from the previous)
```

Telerik Academy

Singleton Pattern

Live Demo

Telerik Academy

Augmenting Modules

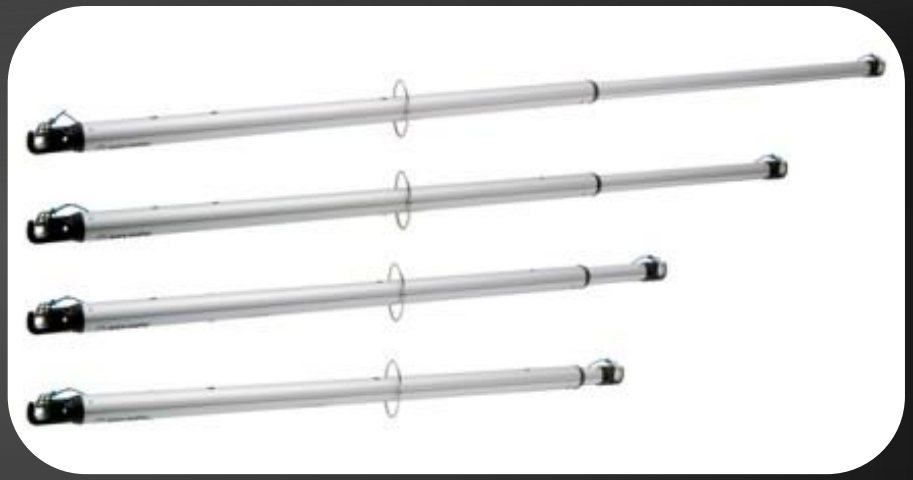
- Augmenting modules means "Split modules in many files/IIFEs:
  - Can be used like a module/revealing module pattern, but with a small fix:
    - module-1.js

```
var module = module || {}; // if module exists
(function(scope){
  scope.obj1 = { /* core for obj1 */ };
}(controls));
```

    - module-2.js

```
var module = module || {}; // if module exists
(function(scope){
  scope.obj2 = { /* core for obj2 */ };
}(controls));
```

Telerik Academy



Augmenting Modules

Live Demo

Telerik Academy

Modules and Patterns

http://academy.telerik.com

Free Trainings @ Telerik Academy

- "C# Programming @ Telerik Academy
  - csharpfundamentals.telerik.com

- Telerik Software Academy
  - academy.telerik.com

- Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy

- Telerik Software Academy Forums
  - forums.academy.telerik.com