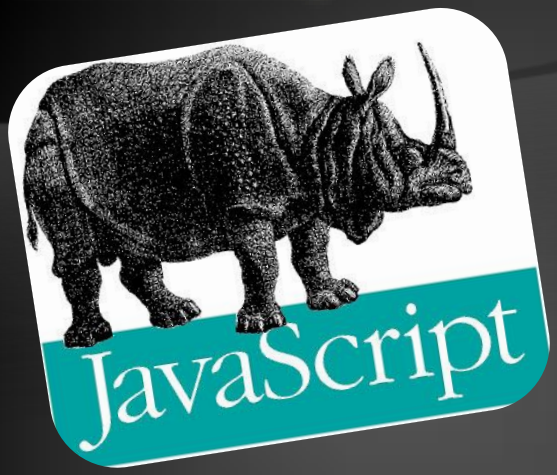
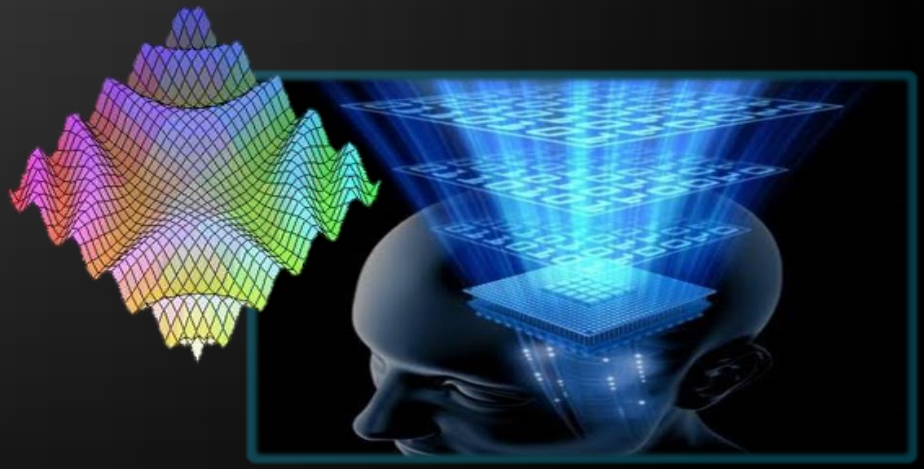
Telerik Academy





# JavaScript Patterns

## Private/Public fields, Module, Revealing Module



**Telerik Software Academy**
Learning & Development Team
http://academy.telerik.com

Telerik Academy

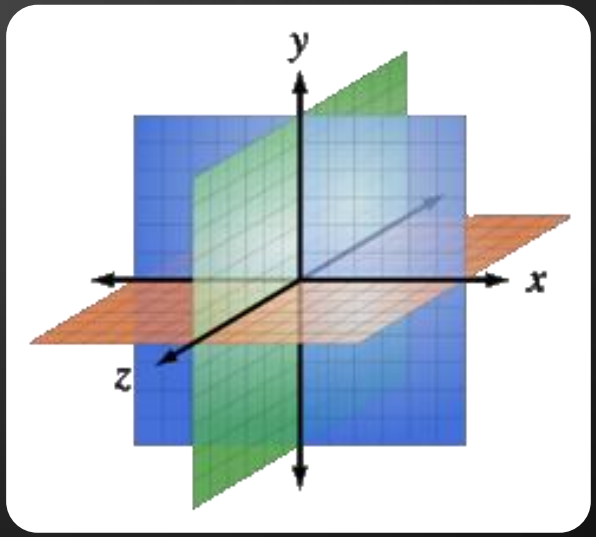
# Table of Contents

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

Telerik Academy

# Public/Private fields

## Using the function scope



# Public/Private Fields

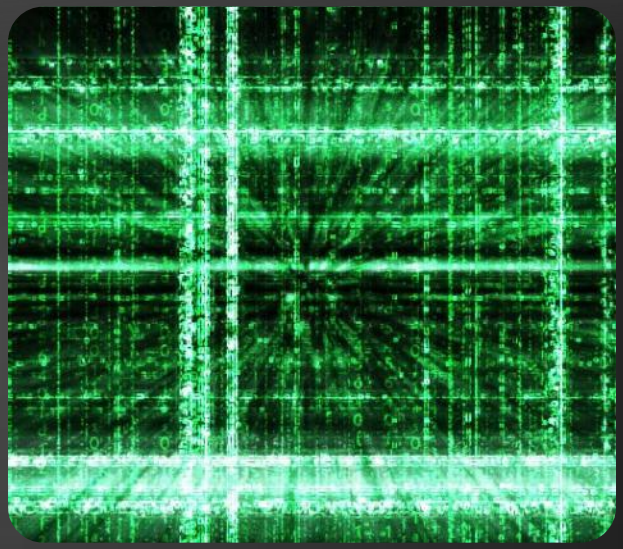
- Each variable is defined:
  - In the global scope (Public)
  - In a function scope (Private)

```
var global = 5;

function myFunction() {
	var private = global;

	function innerFunction(){
		var innerPrivate = private;
	}
}
```

Telerik Academy



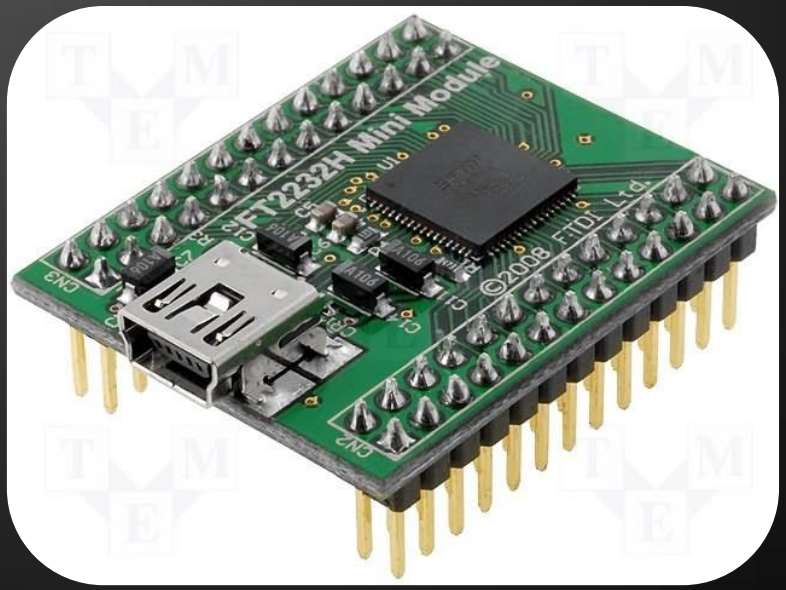
# Public/Private fields

Live Demo

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 () {
  //hidden members
  function formatResult(name, value) { //… }

  //visible members
  return {
    Calculator: function (name) {
      var result;
      result = 0;
      this.add = function (x) { //… };
      this.subtract = function (x) { //…};
      this.showResult = function () { //… };
    }
  };
}());
var calc = new controls.Calculator('First');
calc.add(7);
calc.showResult();
```

# Module Pattern: Example

- Example:

```
var controls = (function () {
  //hidden members
  function formatResult(name, value) { //… }

  //visible members
  return {
    Calculator: function (name) {
      var result;
      result = 0;
      this.add = function (x) { //… };
      this.subtract = function (x) { //…};
      this.showResult = function () { //… };
    }
  };
}());
var calc = new controls.Calculator('First');
calc.add(7);
calc.showResult();
```

The visible members create closures with them

# Module Pattern: Example

- Example:

```
var controls = (function () {
  //hidden members
  function formatResult(name, value) { //… }

  //visible members
  return {
    Calculator: function (name) {
      var result;
      result = 0;
      this.add = function (x) { //… };
      this.subtract = function (x) { //…};
      this.showResult = function () { //… };
    }
  };
}());
var calc = new controls.Calculator('First');
calc.add(7);
calc.showResult();
```

Visible function constructor

# 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

# 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 () {
  //hidden function
  function formatResult(name, value) { // … }
  var Calculator = (function () {
    var Calculator = function (name) { // … };
    Calculator.prototype.add = function (x) { // … };
    Calculator.prototype.subtract = function (x) { // … };
    Calculator.prototype.showResult = function () { // …};
    return Calculator;
  }());
  return {
    Calculator: Calculator
  };
}());
var calc = new controls.Calculator('First');
```

# Revealing Module Pattern: Example

- Example:

```
var controls = (function () {
  //hidden function
  function formatResult(name, value) { // … }
  var Calculator = (function () {
    var Calculator = function (name) { // … };
    Calculator.prototype.add = function (x) { // … };
    Calculator.prototype.subtract = function (x) { // … };
    Calculator.prototype.showResult = function () { // …};
    return Calculator;
  }());
  return {
    Calculator: Calculator
  };
}());
var calc = new controls.Calculator('First');
```

Create the function constructor hidden

# Revealing Module Pattern: Example

- Example:

```
var controls = (function () {
  //hidden function
  function formatResult(name, value) { // … }
  var Calculator = (function () {
    var Calculator = function (name) { // … };
    Calculator.prototype.add = function (x) { // … };
    Calculator.prototype.subtract = function (x) { // … };
    Calculator.prototype.showResult = function () { // …};
    return Calculator;
  }());
  return {
    Calculator: Calculator
  };
}());
var calc = new controls.Calculator('First');
```

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

# The Revealing Prototype Pattern

## Reveal the most interesting members (again)

# Revealing Prototype Pattern: Pros and Cons

- Pros:
  - "Modularize" code into re-useable objects
  - Variables/functions taken out of global namespace
  - Expose only public members
  - Functions are loaded into memory once
  - Extensible
- Cons:
  - "this" can be tricky
  - Constructor is separated from prototype

# Revealing Prototype Pattern: Structure

- Structure:

```
var Constructor = function () {
        //constructor defined here
}

Constructor.prototype = (function() {
        //hidden variables
        //hidden functions

        return {
                //exposed members
                someFunc: pointerToSomeFunc
                anotherFunc: pointerToAnotherFunc
        };
}());
```

# Revealing Prototype Pattern: Example

- Example:

```
var Calculator = function (name) { // … };

Calculator.prototype = (function () {
  var add, subtract, showResult, formatResult;

  add = function (x) { // … };
  subtract = function (x) { // … };
  showResult = function () { // … };
  formatResult = function (name, value) { // … };

  return {
    add: add,
    subtract: subtract,
    showResult: showResult
  };
}());

var calc = new Calculator('First');
```

# Revealing Prototype Pattern: Summary

- Module pattern provides encapsulation of variables and functions
- Provides a way to add visibility (exposed versus hidden) to members
- Provides extension capabilities

Telerik Academy

# Revealing Prototype Pattern

Live Demo

Telerik Academy

# Singleton Pattern

## One object to rule them all!

# Singleton Pattern: Structure

- Structure:

```
var module = function() {
  var instance, getInstance;

  return {
    getInstance: function(){
      if(!instance){
        instance = new Instance();
      }
      return instance;
    }
  };
}();
```

# Singleton Pattern: Example

- Example:

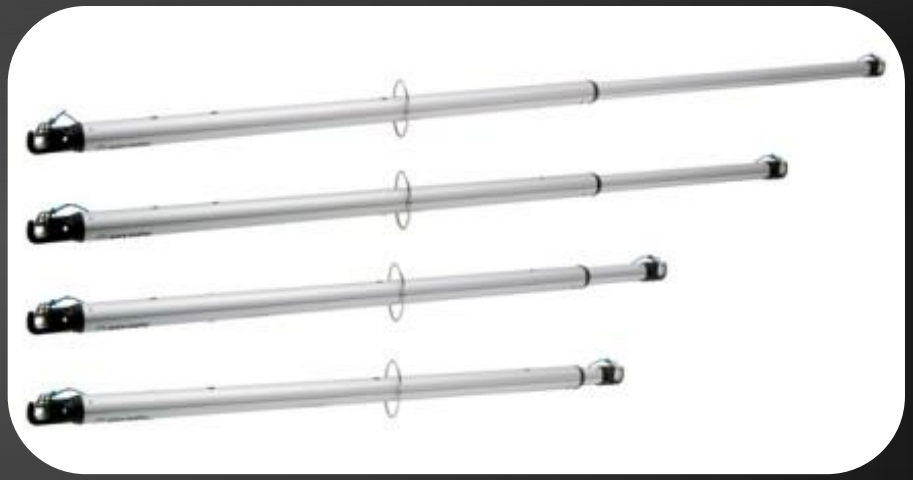
```
var controls = function () {
  var Calculator, calculatorInstance;
  Calculator = (function () {
    function Calculator() { //… }
    return Calculator;
  }());
  return {
    getCalculator: function () {
      if (!calculatorInstance) {
        calculatorInstance = new Calculator();
      }
      return calculatorInstance;
    }
  };
}();
var calculator = controls.getCalculator();
```

Telerik Academy

# Singleton Pattern

## Live Demo

Telerik Academy



# Augmenting Modules

## Live Demo

Telerik Academy

# Arrays

# Questions?

http://csharpfundamentals.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

# Homework

Create the Snake game using the Revealing module pattern. Design the game such that it has at least three modules.