# JavaScript Introduction

Topics discussed this presentation

- Scope
- Closure

# Scope

## Global variables

Variables declared **inside function**

- Visible throughout function
- Invisible outside function

Variables declared **outside function**

- These are **global** variables
- Content both files visible program-wide.
- Avoid using global variables

```
//file: script1.js
velocity= 10;

//file: script2.js
function square(x){
  return x * x;
}

//velocity visible here because global
console.log(square(velocity)); // 100
```

# Scope

## Global variables

Code below left generates error in *strict* mode:

- Uncaught ReferenceError: velocity is not defined

```
// Invalid code: undeclared global
'use strict';
velocity= 10;
```

```
// Valid code: declared global
'use strict';
let velocity = 10;
const speed = 10;
var acceleration = 10;
```

# Scope

## Global variables

Variables defined but not declared **inside function**

- Are global variables
- Referred to as *implied global*
- Dangerous practice - avoid
- Use ES6 strict disallows

```
let circle;
function requestReport(){
    center = circle.getCenter();
}
```

```
circle is a global variable
center is (implied) global variable
```

# Scope

## Implied Globals

Defined but not declared in function

- `velocity` is implied global
- Visible program-wide once f() invoked
- Alert box displays 100
- Illegal in strict mode
- ReferenceError: velocity is not defined

```
function f() {
  velocity= 100;
}

f();
alert(velocity);
```

# Scope

## Function scope

Defined but not declared in function

- `velocity` is implied global
- Visible program-wide once f() invoked
- Alert box displays 100
- Illegal in strict mode
- ReferenceError: velocity is not defined

```
function f() {
  velocity= 100;
}

f();
alert(velocity);
```

# JavaScript

## Global Object

In the browser, the global object is the **window** object

javascript.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>JavaScript</title>
  </head>
  <body>
      <script src="javascript1.js"></script>
      <script src="javascript2.js"></script>
  </body>
</html>
```

2 script files loaded into global space

javascript1.js

```
var cars = [ 'Ford', 'Honda', 'Nissan', 'Peugot' ];

var functionOne = function (x) {
    console.log(x);
};

functionOne (this);
```

javascript2.js

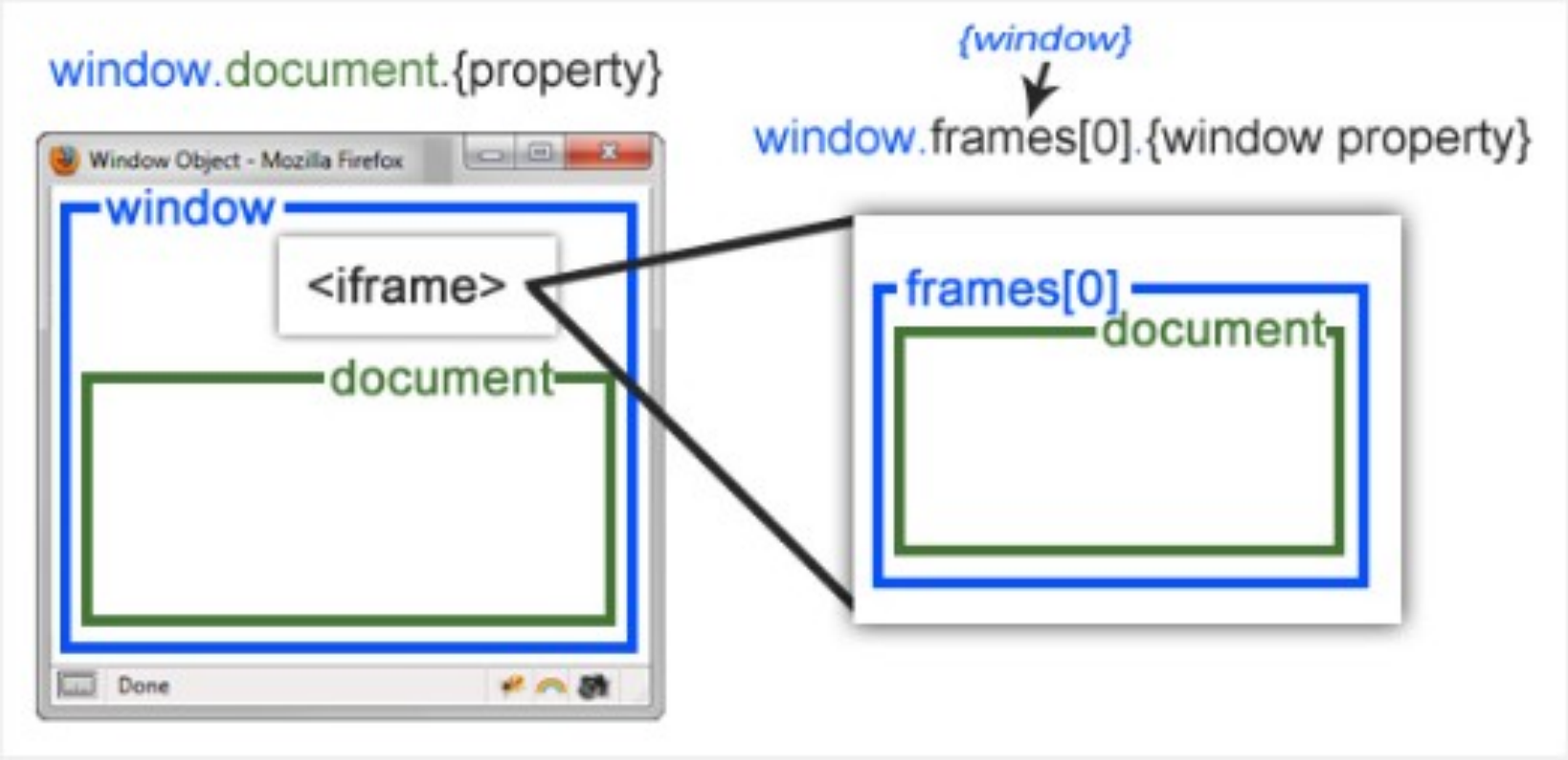
```
var trucks = [ 'Volvo', 'Saab', 'Mercedes'];

var functionTwo = function (x) {
    console.log(x);
};

functionTwo(this);
```

Both arguments **x** are **this** reference to Window object
Observe that contents of both files loaded into same global space.
This is a major weakness in JavaScript design.

# JavaScript

## Global Object

### Window and document objects




Javascript Window Object with Iframe

# JavaScript

## Global Abatement

Define global variable for app

- const `MyApp={}`
- This becomes container for app

```
const MYAPP = {};

MYAPP.square = function (x) {
  return x * x;
};

console.log(MYAPP.square(val));
```

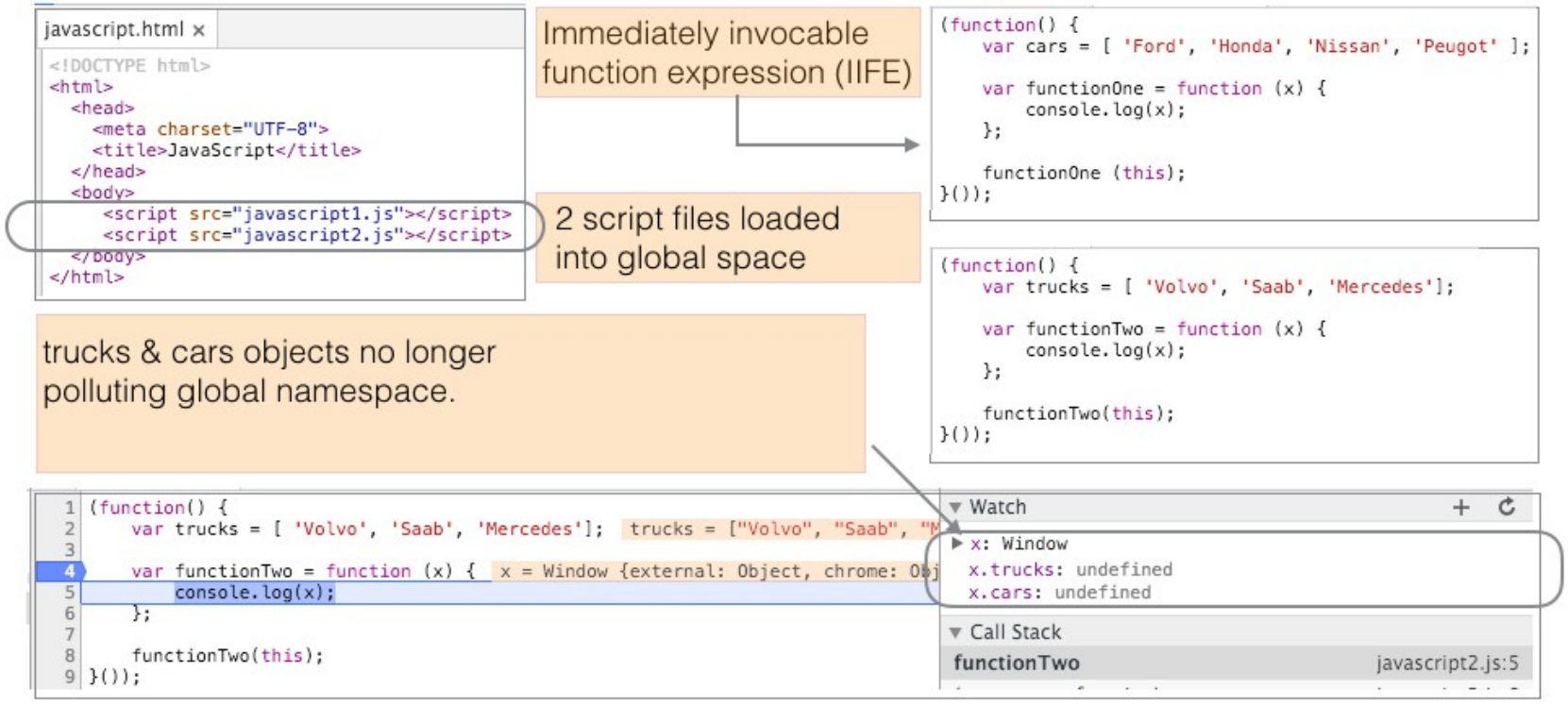
# Global Abatement

## Using Immediately Invoked Function Expression (IIFE)

```
(function() {
// code here
// objects declared here not visible outside function
}());
```

# Global Abatement

## Using Immediately Invoked Function Expression (IIFE)



# Global Abatement

## IIFE pattern used for global abatement

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='UTF-8'>
    <title>AbateGlobals</title>
  </head>
  <body>
    <button type='button' onclick='clickMe()'>ClickMe</button>
    <script src='abateglobals.js'></script>
  </body>
</html>
```

# Global Abatement

## IIFE pattern used for global abatement

Function *clickMe* invoked on button press:

```
(function (context){
  const bar = 100;
  context.clickMe = function () {
    foo();
    alert('hey, it\'s me');
  };

  function foo() {
    alert('in foo');
  }
}());
```

# jQuery

## IIFE and document ready interaction

```
(function () { // <= IIFE
  // Do something that doesn't require DOM to be ready.
  console.log('within IIFE');
  $(function () { // <= same as $(document).ready(function () {
    // Do something involving DOM manipulation.
    console.log($('#p'));
  });
}());
```

# Scope

## Global abatement

```
// myapp.js
MYAPP = (function () {

  function square(x){
    return x * x;
  }

  function cube(x){
    return x * square(x);
  }

  return {
    square,
    cube,
  };
}());
```

```
<!DOCTYPE html>
<html>
<head>
...
</head>
<body>
  <script src="myapp.js"></script>
  <script src="calculator.js"></script>
</body>
</html>
```

```
// calculator.js
let x2 = MYAPP.square(10); // 100
let x3 = MYAPP.cube(10); // 1000
```

# Java

## Block Scope

Java has *block scope*:

- Variable **y** out of scope (invisible) outside its block

```
public void scope() {
  int x = 10;
  {
    int y = 100;
  }
  System.out.println('x : ' + x); // ok: x in scope
  System.out.println('y : ' + y); // compile-time error: y out of scope
}
```

# JavaScript function scope

## Hoisting

- Declaration of **y** *hoisted* to top of function.
- Initialization of **y** takes place as shown.

```
function scopeExample()
{
  console.log('y : ', y); // undefined
  var x = 10;
  {
    var y = 100;
  }

  console.log('x : ', x); // 10
  console.log('y : ', y); // 100
};

scopeExample();
```

# JavaScript block scope

## ES6 variables **let** and **const**

- *var* replaced by *let*
- ReferenceError: y is not defined
- *let* and *const* have block scope.

```
function scopeExample()
{
  console.log('y : ', y); // ReferenceError
  let x = 10;
  {
    let y = 100;
  }

  console.log('x : ', x); // 10
  console.log('y : ', y); // 100
};

scopeExample();
```

# JavaScript block scope

## Test your knowledge

- What is the console output here?

```
var x = 'outer scope';

function f() {  console.log('x : ', x); var x = 'inner
  scope';
}

f();
```

# JavaScript block scope

## Test your knowledge

- One change has been made: *var x = . . .* commented out.
- What is the console output now?

```
var x = 'outer scope';

function f() {
  console.log('x : ', x);
  // var x = 'inner scope';
}

f();
```

# JavaScript block scope

## Test your knowledge

- Final change: replace *var x* with *let*.
- How does this influence the output?

```
var x = 'outer scope';

function f() {
  console.log('x : ', x);
  let x = 'inner scope';
}

f();
```

# Scope

## Temporal dead zone (TDZ)

- Already encountered TDZ above.
- **let** and **const** are hoisted.
- Values **undefined** until initialized.
- In meantime, are in TDZ:
  - Attempted access before initialization generates error
  - ReferenceError: x is not defined

```
let x = 'outer scope';
function f() {
  console.log('x : ', x); // <= x in TDZ, value undefined
  let x = 'inner scope';
}

f();
```

# Scope

## **this** variable

```
let calculator = {
  result: 0,
  multiply: function (number, multiplier){
    let _this = this; // this is bound to calculator object
    let helper = function (x, y) {
      _this.result = x * y; // this is bound to global object
    };

    helper(number, multiplier);
    return _this;
  },

  getResult: function () {
    return this.result;
  },
};

console.log(calculator.multiply(9, 4).result); // => 36
```

# Scope

## **this** variable problem solved by using arrow function

```
let calculator = {
  result: 0,
  multiply: (number, multiplier) => {
    let helper = function (x, y) {
      this.result = x * y;
    };

    helper(number, multiplier);
    return this;
  },

  getResult: function () {
    return this.result;
  },
};

console.log(calculator.multiply(9, 4).result); // => 36
```

# JavaScript Closure

## A Powerful Feature

An inner function that has access to

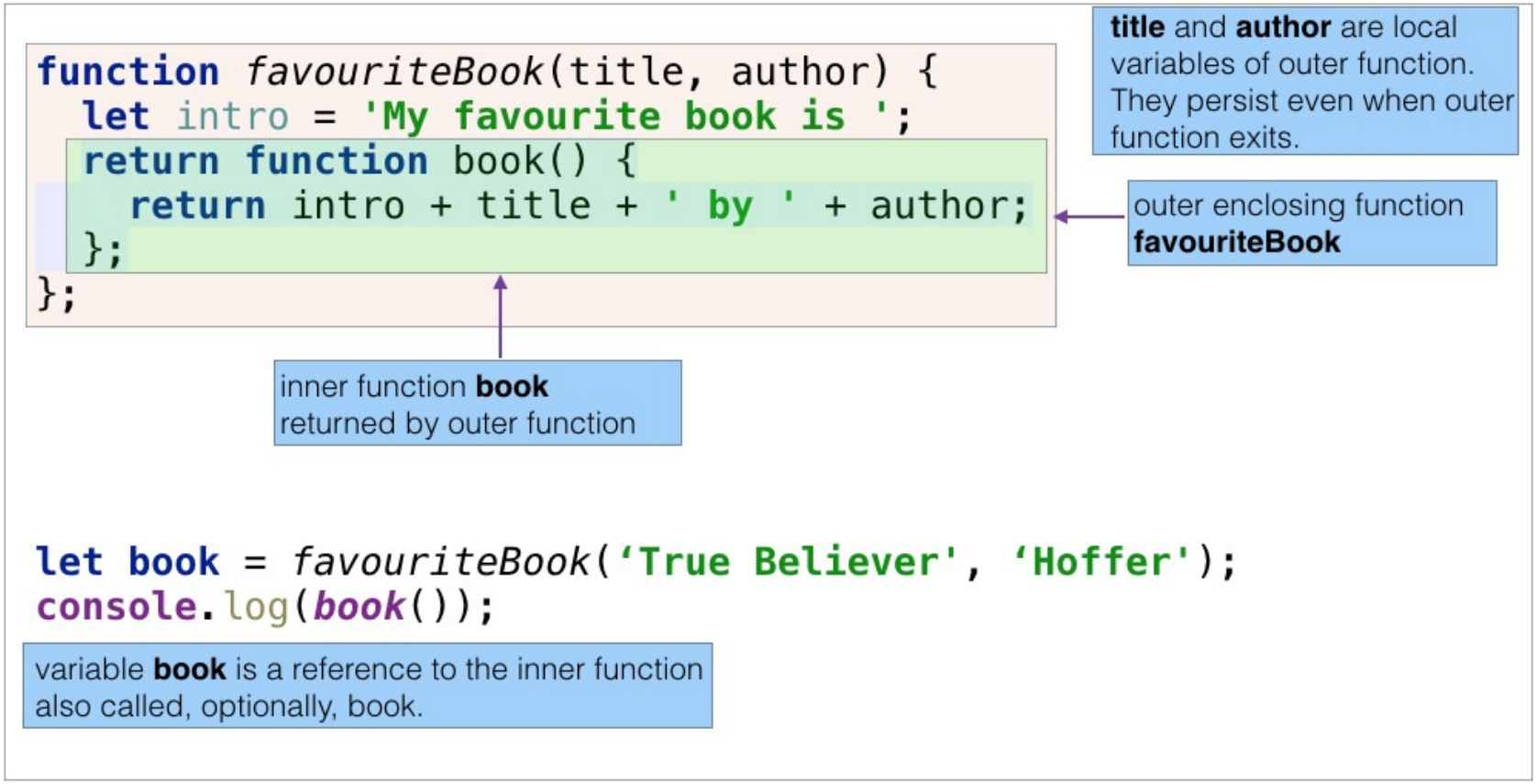
- its own variables,
- the outer enclosing function's variables,
- the global variables.

**This holds even when outer function has returned.**

```
function favouriteBook(title, author){
  const intro = 'My favourite book is ';
  return function book(){
    return intro + title + ' by ' + author;
  };
};

const favourite = favouriteBook('Eloquent JavaScript', 'Marijn Haverbeke');
console.log(favourite());
```

# JavaScript Closure

## A Powerful Feature



# JavaScript Closure

## A Powerful Feature

### A closure example from Eloquent JavaScript

```
// @see page 50 http://eloquentjavascript.net/Eloquent_JavaScript.pdf
function multiplier(factor){
  return function (number){
    return number * factor;
  };
}

const twice = multiplier(2);
const result = twice(5);
console.log(result); // => 10

const thrice = multiplier(3);
result = thrice(5);
console.log(result); // => 15
```

# JavaScript Closure

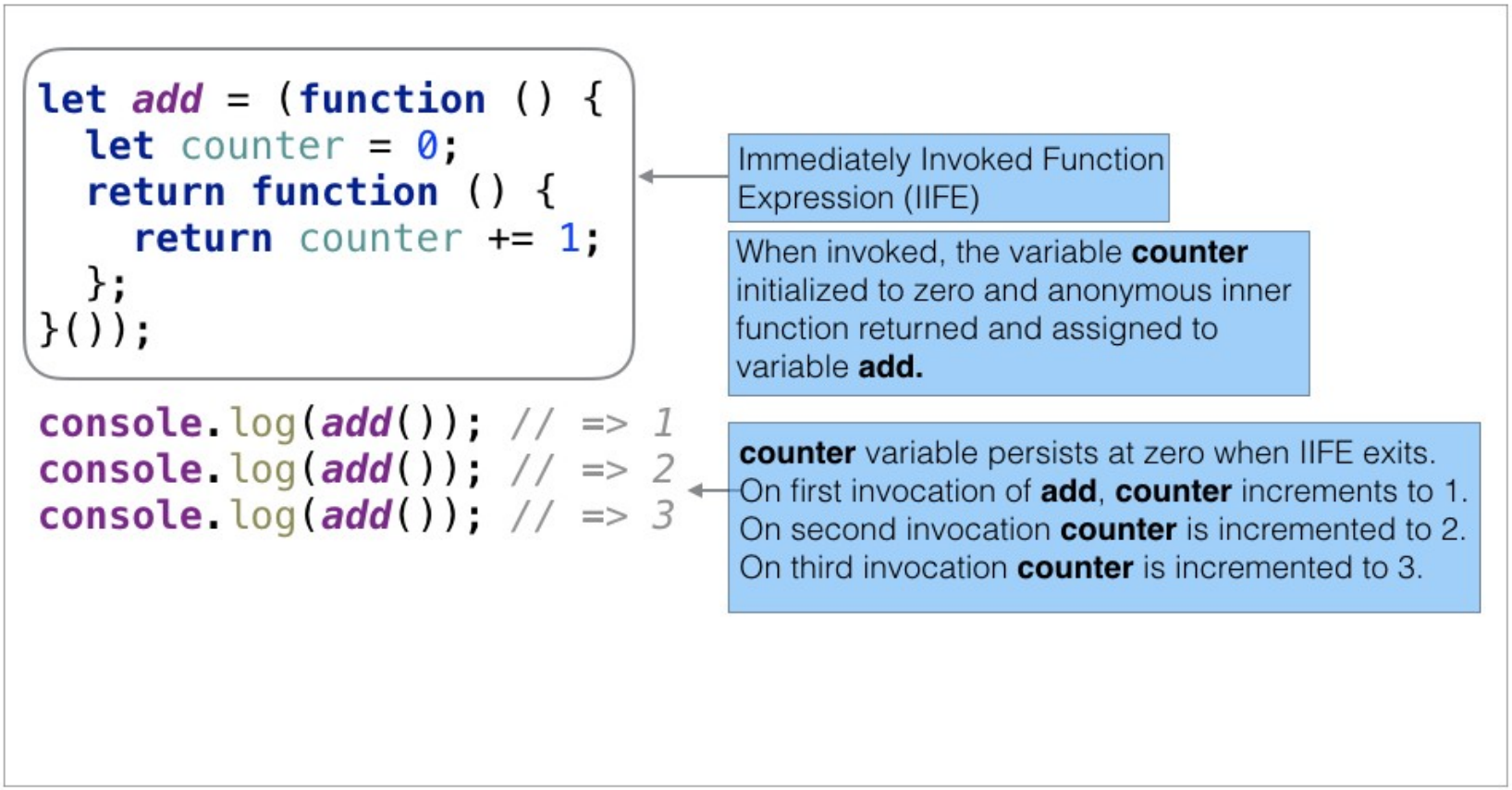
## A Powerful Feature

### A closure example from w3schools

```
// @see http://www.w3schools.com/js/js_function_closures.asp
const add = (function () {
  let counter = 0;
  return function () {
    return counter += 1;
  };
}());

console.log(add()); // => 1
console.log(add()); // => 2
console.log(add()); // => 3
```

# JavaScript Closure

## A Powerful Feature



# JavaScript Closure

## Final Example

```
myObject =
(function() {
   let value = 0;
   return {
      increment : function(inc) {
         value += typeof inc === 'number' ? inc : 1;
      },
      getValue : function() {
         return value;
      }
   };
}());

myObject.increment();
console.log(myObject.getValue()); // => 1
myObject.increment(2);
console.log(myObject.getValue()); // => 3
```

Immediately invocable function expression (IIFE)

return value is object

# JavaScript

## Presentation summary

- Globals
  - Avoid use global variables.
  - Avoid polluting global namespace.
  - Use global abatement technique(s).
- Scope
  - Pre ES6 - only function scope.
  - ES6 adds block scope.
- Closure
  - A powerful language feature.