Web Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology

http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

JavaScript

JAVASCRIPT FUNDAMENTALS

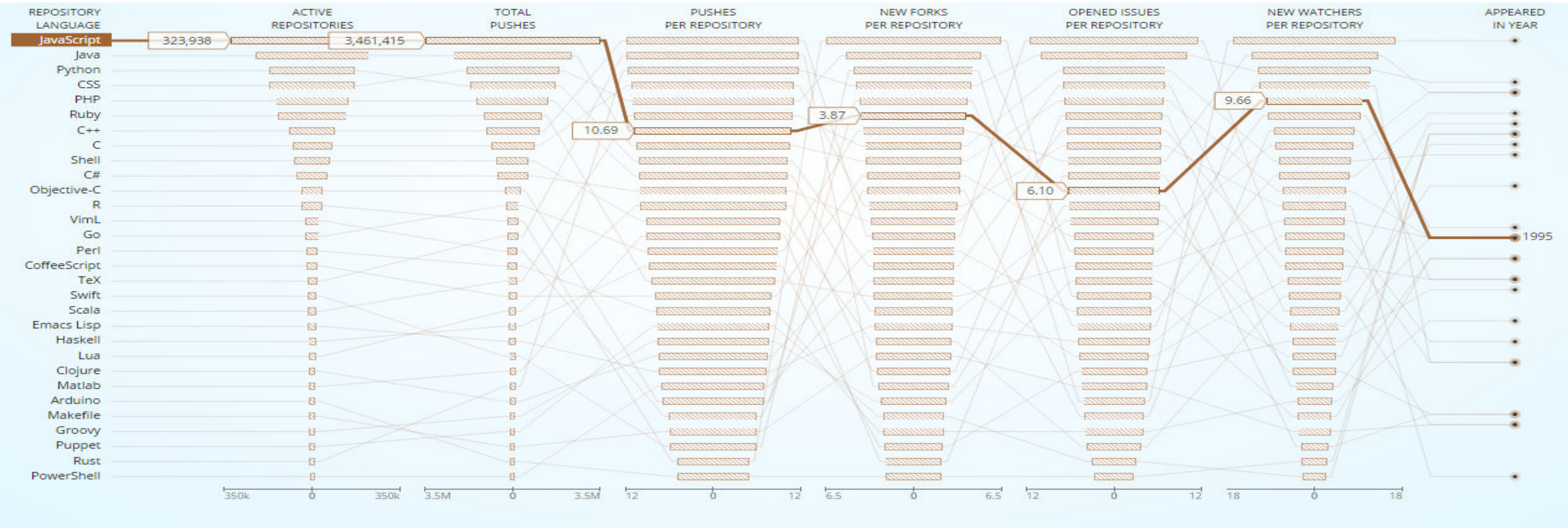
Agenda

Data Types, Objects & Arrays

Looping & Iteration

Functions, Methods & Constructors

Top Languages on GitHub (19/02/2015)



JavaScript Data Types

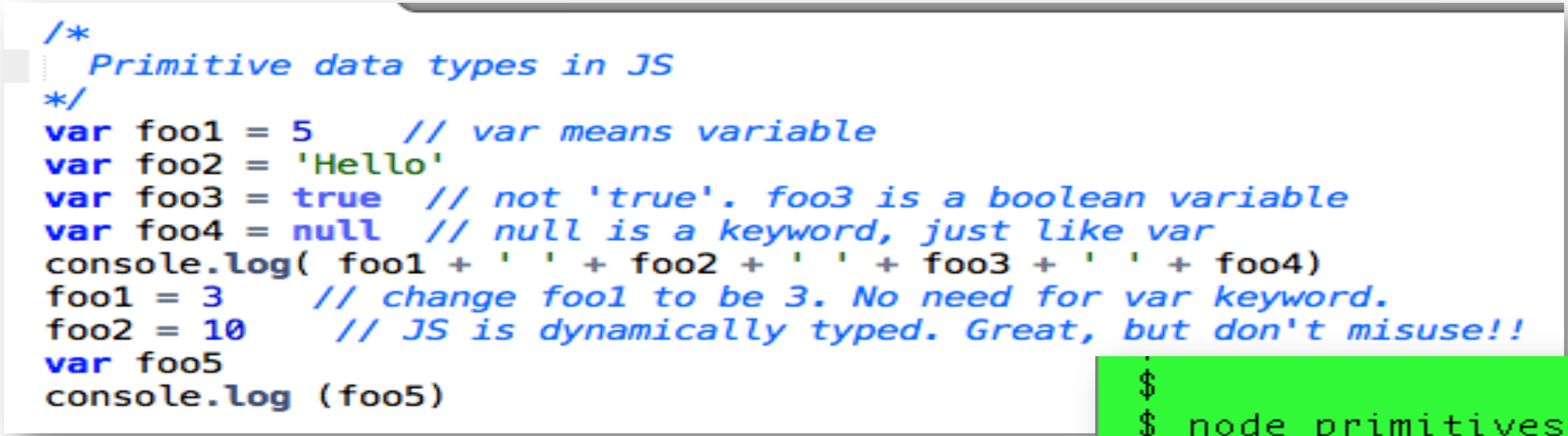
Language data types:

1. Primitives: number, string, boolean, null, undefined.
2. Everything else is an object (even functions).

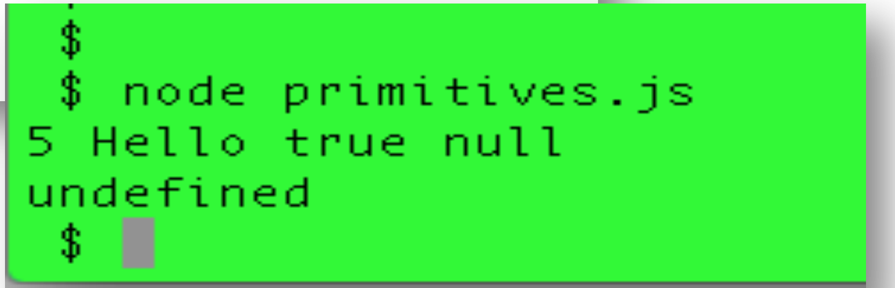
JS is a dynamically typed language.

Primitive types

Suppose this code is in a file, called *primitives.js*



```
/*
  Primitive data types in JS
*/
var foo1 = 5     // var means variable
var foo2 = 'Hello'
var foo3 = true   // not 'true'. foo3 is a boolean variable
var foo4 = null   // null is a keyword, just like var
console.log( foo1 + ' ' + foo2 + ' ' + foo3 + ' ' + foo4)
foo1 = 3     // change foo1 to be 3. No need for var keyword.
foo2 = 10     // JS is dynamically typed. Great, but don't misuse!!
var foo5
console.log (foo5)
```



```
 $
 $ node primitives.js
5 Hello true null
undefined
 $
```

Thanks to the node.js platform,

I can execute this code from the command line – no browser needed.

Primitive types (The syntax)

var foo = 20

var – keyword to indicate we are declaring something – a primitive number variable in this case.

Identifier – ‘**foo**’ is an identifier or name for this thing.

- Lots of rules about valid format for identifiers (no spaces, don’t start with numeric character, etc etc)

Operator **– +, =, *** (multiply)**, –, [ ]** (subscript) etc

- Some rules about where they can appear in a statement.

Objects

The object - fundamental structure for representing complex data.

A unit of composition for data (or STATE).

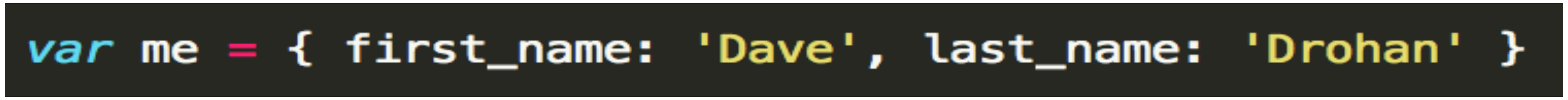
Objects are a set of key-value pairs defining properties.

- Keys (property names) are identifiers and must be unique
- Values can be any type, including other objects (nesting).

Literal syntax for defining an object:

{ <key1> : <value1>, <key2> : <value2>, ...}

- Example:



```
var me = { first_name: 'Dave', last_name: 'Drohan' }
```

Objects

Two notations for accessing the value of a property:

1. Dot notation e.g me.first_name
2. Subscript notation e.g. me['first_name']   (Note quotes)

Same notations for changing a property value.

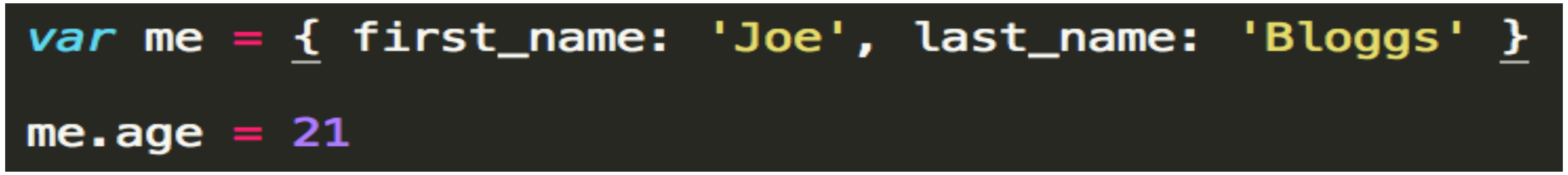
```
me.first_name = 'Joe'
me[ ‘last_name’ ] = 'Bloggs'
```

Subscript notation allows the subscript be a variable reference.

```
var foo = 'last_name'
me[foo] = ……..
```

Objects are dynamic

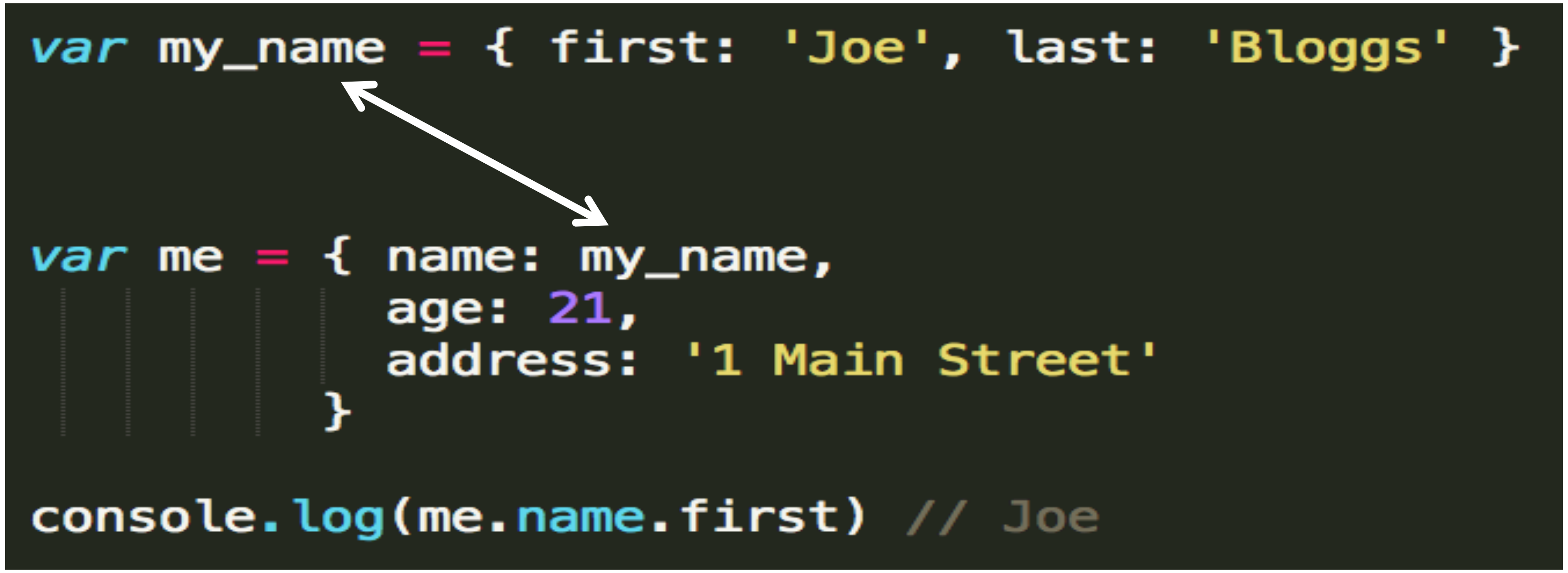
Properties can be added and removed at any time – JS is dynamic.



```
var me = { first_name: 'Joe', last_name: 'Bloggs' }

me.age = 21
```

Object property

A property value can be a variable reference.



```
var my_name = { first: 'Joe', last: 'Bloggs' }


var me = { name: my_name,
           age: 21,
           address: '1 Main Street'
         }

console.log(me.name.first) // Joe
```

Object keys

Internally JS stores keys as strings.

Hence the subscript notation – me[‘age’]

Array Data Structure

Dfn: Arrays are an ordered list of values.

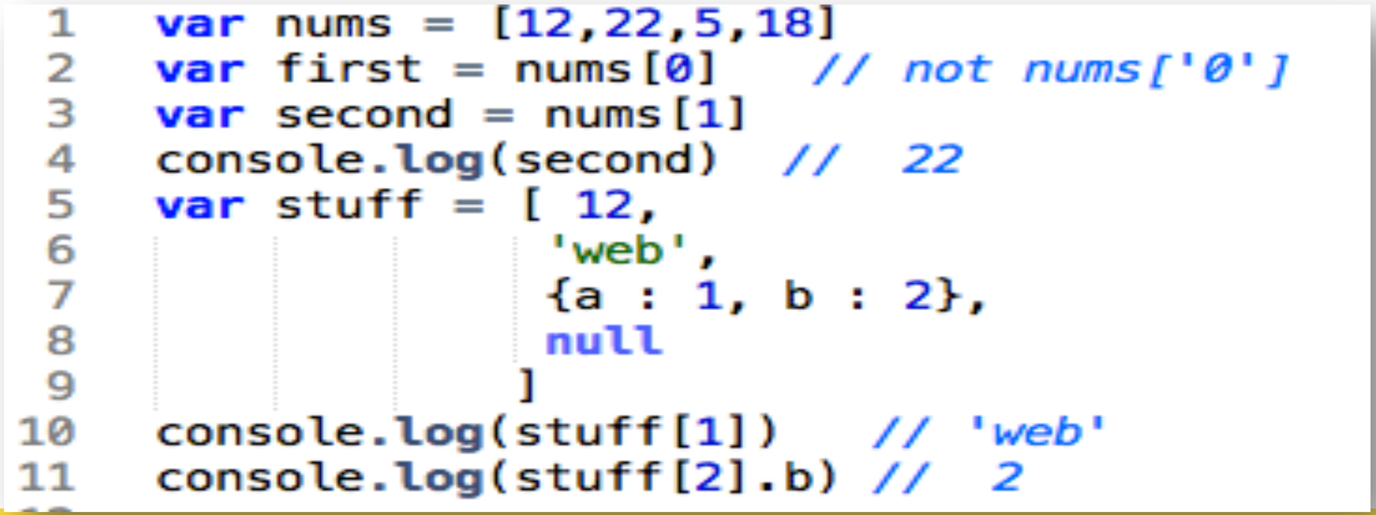
- An object's properties are not ordered.

Literal syntax: [<value1>,<value2>,...]

In JS, the array values may be of mixed type.

- Although mixed types may reflect bad design.

Use an index number with the subscript notation to access individual elements:



```
var nums = [12,22,5,18]
var first = nums[0]    // not nums['0']
var second = nums[1]
console.log(second)  //  22
var stuff = [ 12,
              'web',
              {a : 1, b : 2},
              null
            ]
console.log(stuff[1])    // 'web'
console.log(stuff[2].b) //  2
```

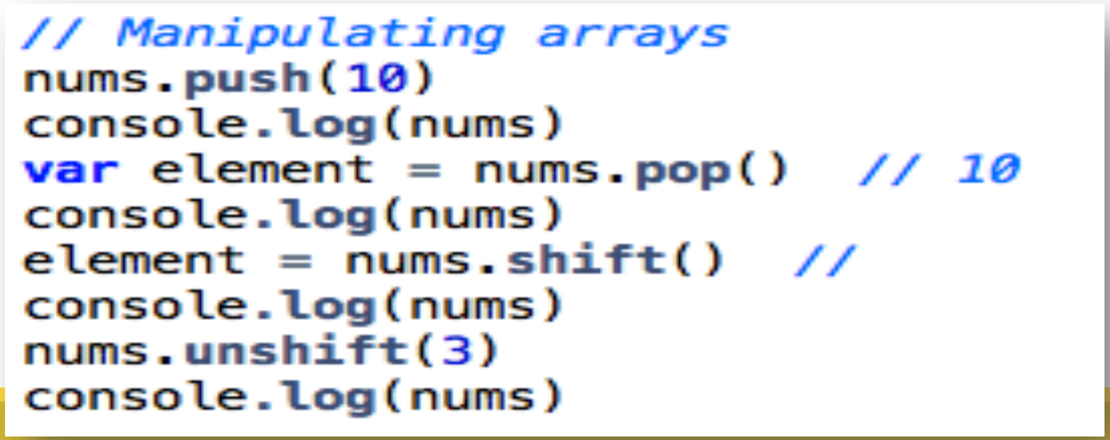
Array Data Structure

In JS, arrays are really just ‘special’ objects:

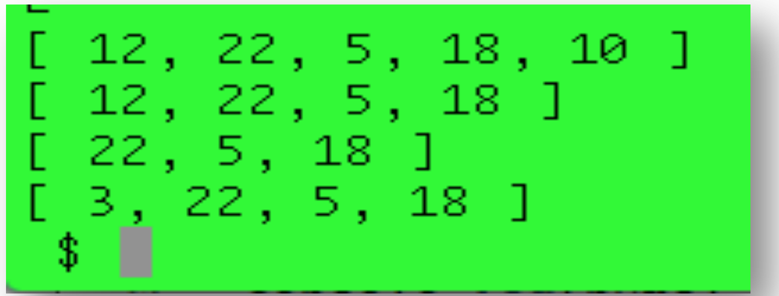
- The indexes are not numbers, but properties - the index number is converted into a string:

  nums[‘2’] same as nums[2]

- Special length property, e.g. var len = nums.length // 4
- Some utility methods for manipulating elements e.g push, pop, shift, unshift, join etc
  - push/pop – add/remove at the tail.
  - shift/unshift – add/remove at the head.

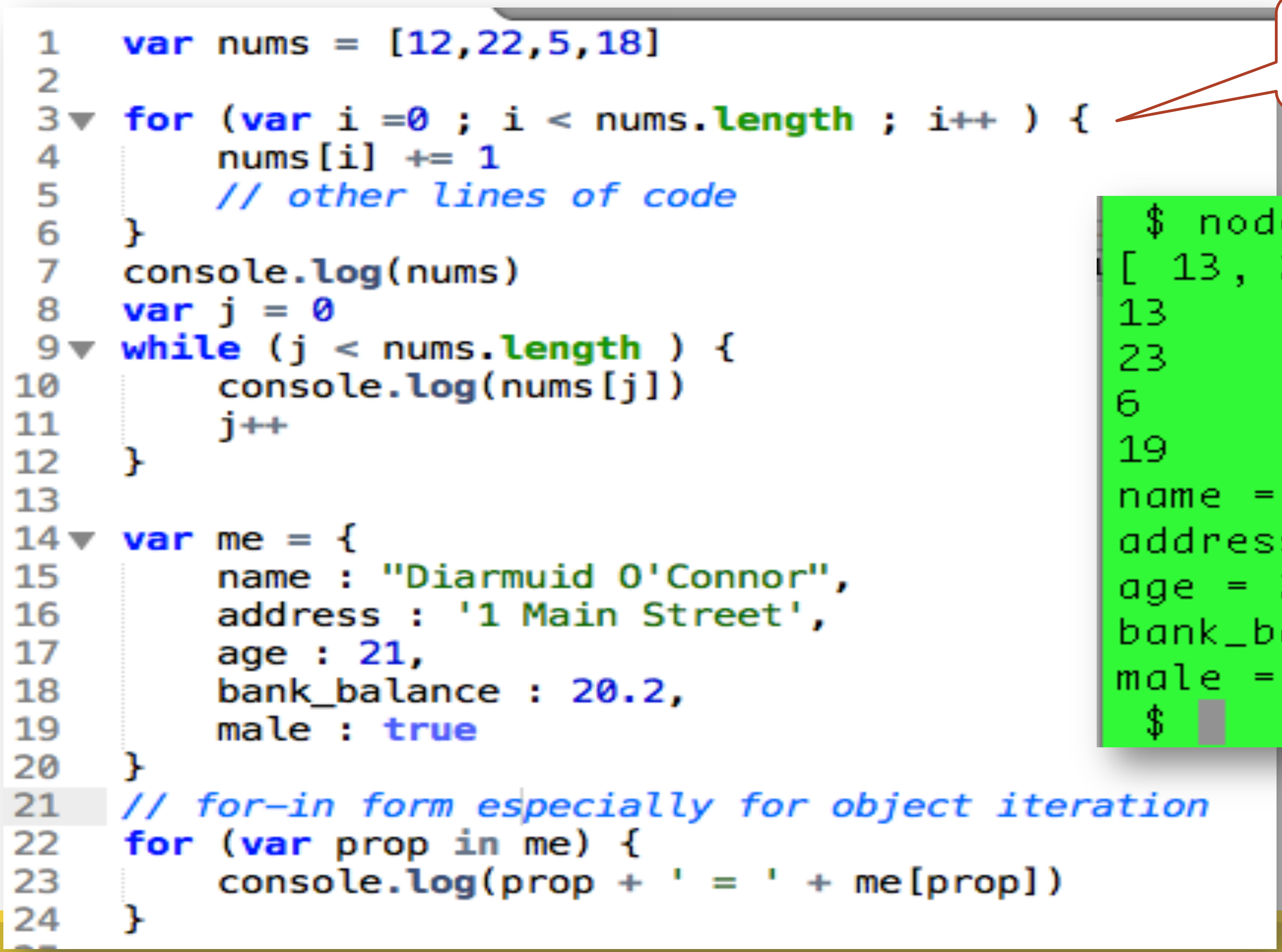


```
// Manipulating arrays
nums.push(10)
console.log(nums)
var element = nums.pop()  // 10
console.log(nums)
element = nums.shift()  //
console.log(nums)
nums.unshift(3)
console.log(nums)
```



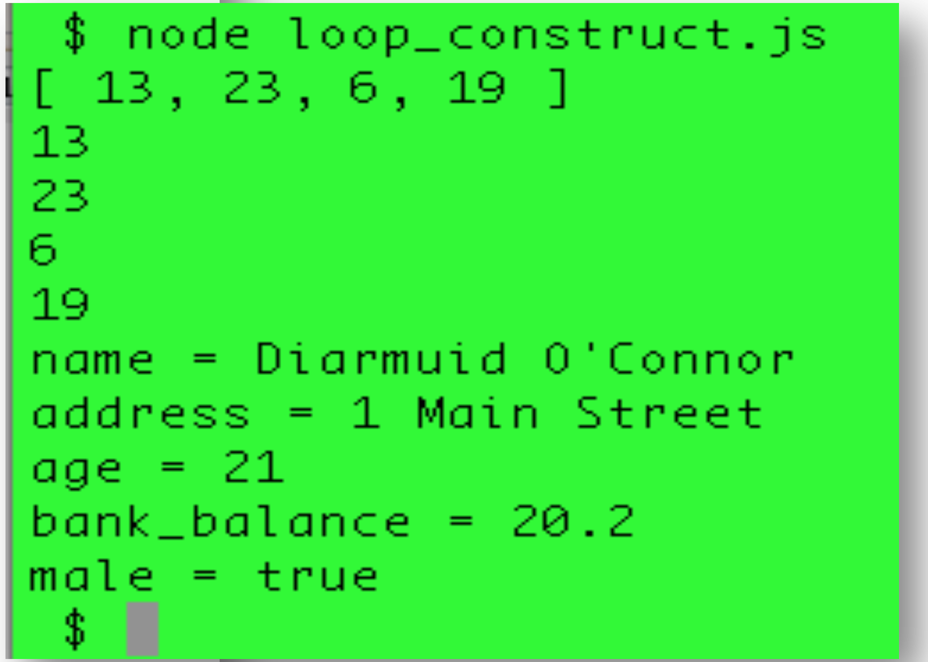
```
[ 12, 22, 5, 18, 10 ]
[ 12, 22, 5, 18 ]
[ 22, 5, 18 ]
[ 3, 22, 5, 18 ]
 $
```

Looping/iteration constructs



```javascript
var nums = [12,22,5,18]

for (var i =0 ; i < nums.length ; i++ ) {
      nums[i] += 1
      // other lines of code
}
console.log(nums)
var j = 0
while (j < nums.length ) {
      console.log(nums[j])
      j++
}

var me = {
      name : "Diarmuid O'Connor",
      address : '1 Main Street',
      age : 21,
      bank_balance : 20.2,
      male : true
}
// for-in form especially for object iteration
for (var prop in me) {
      console.log(prop + ' = ' + me[prop])
}
```

A more elegant form later.



```
 $ node loop_construct.js
[ 13, 23, 6, 19 ]
13
23
6
19
name = Diarmuid O'Connor
address = 1 Main Street
age = 21
bank_balance = 20.2
male = true
 $
```

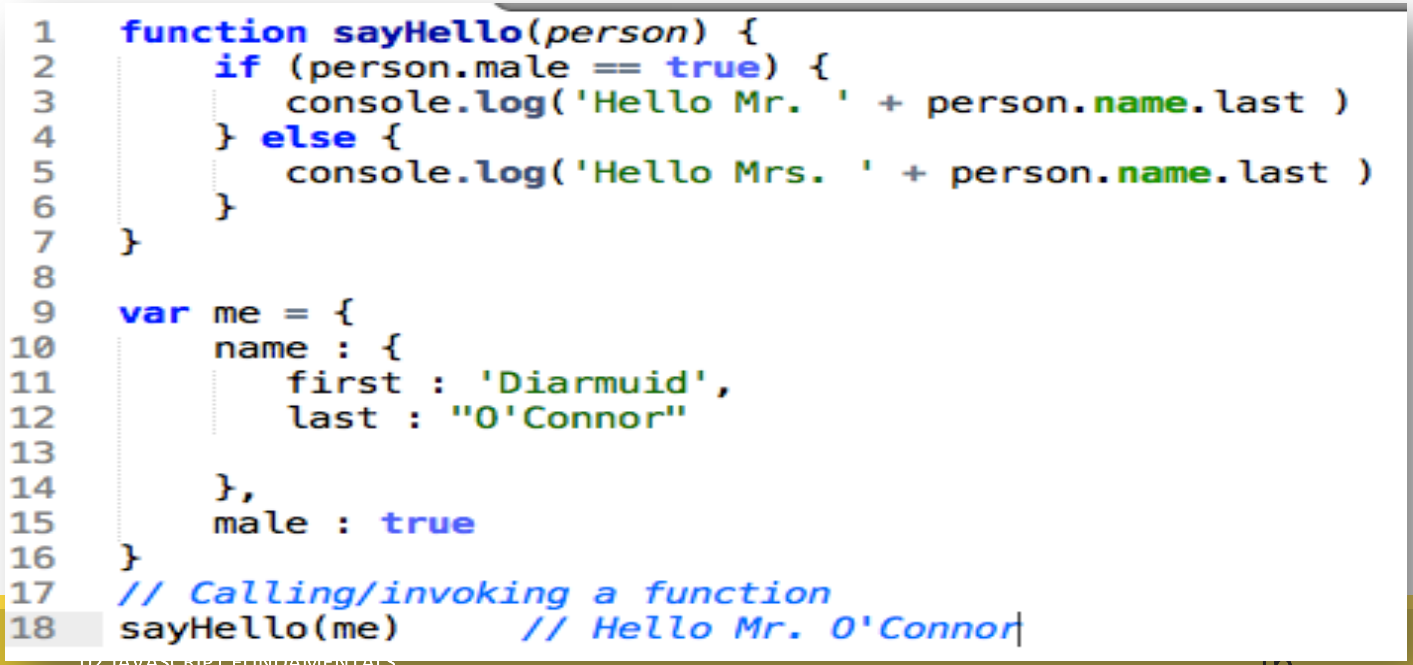
JavaScript functions

Fundamental unit of composition for logic (or BEHAVIOUR).

Basic syntax: function <func_name>(<parameters>) { <body of code> }

- Some functions don’t need parameters.

A function’s body is executed by calling/invoking it with arguments - <func_name>(<arguments>)



```
function sayHello(person) {
    if (person.male == true) {
        console.log('Hello Mr.  ' + person.name.last )
    } else {
        console.log('Hello Mrs.  ' + person.name.last )
    }
}

var me = {
    name : {
        first : 'Diarmuid',
        last : "O'Connor"

    },
    male : true
}
// Calling/invoking a function
sayHello(me)      // Hello Mr. O'Connor
```

Functions - Variable scopes

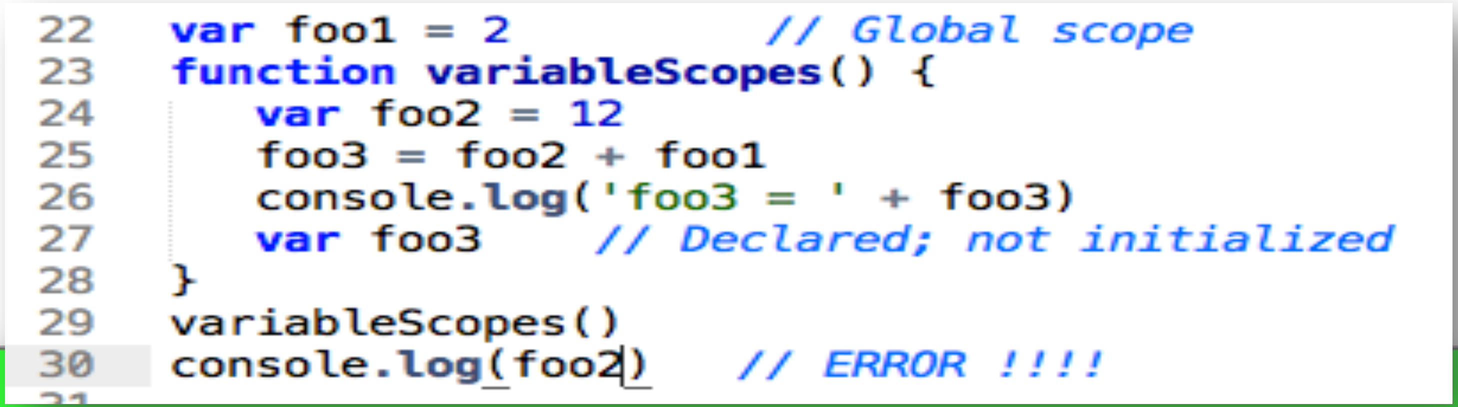
Every function creates a new variable scope.

- Variables declared inside the function are not accessible outside it.
- All variables defined within the function are "hoisted" to the start of the function, as if all the var statements were written first.
  - You can use a variable inside a function before declaring it.

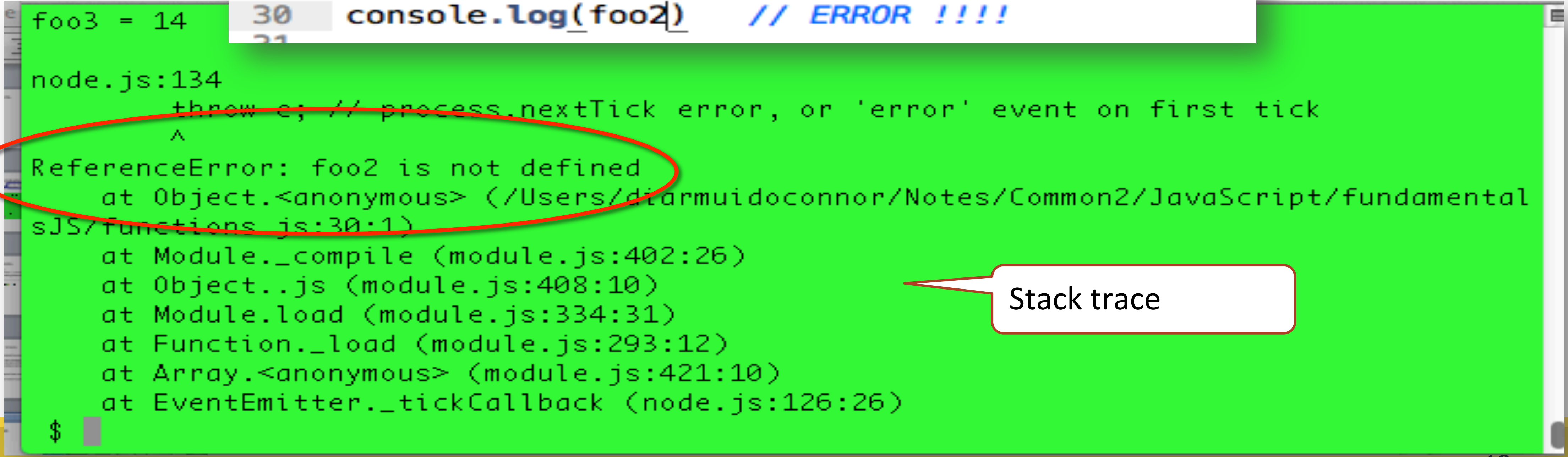
Global scope – default scope for everything declared outside a function's scope.

- Variables in global scope are accessible inside functions.

Functions - Variable scopes



```
var foo1 = 2              // Global scope
function variableScopes() {
    var foo2 = 12
    foo3 = foo2 + foo1
    console.log('foo3 = ' + foo3)
    var foo3      // Declared; not initialized
}
variableScopes()
console.log(foo2)    // ERROR !!!!
```



```
foo3 = 14

node.js:134
        throw e; // process.nextTick error, or 'error' event on first tick
        ^
ReferenceError: foo2 is not defined
    at Object.<anonymous> (/Users/diarmuidoconnor/Notes/Common2/JavaScript/fundamental
sJS/functions.js:30:1)
    at Module._compile (module.js:402:26)
    at Object..js (module.js:408:10)
    at Module.load (module.js:334:31)
    at Function._load (module.js:293:12)
    at Array.<anonymous> (module.js:421:10)
    at EventEmitter._tickCallback (node.js:126:26)
 $
```

Stack trace

JavaScript functions

Can be **created** using:

- ❑ A declaration (previous examples).
- ❑ An expression.
- ❑ A method (of a custom object).
- ❑ An anonymous unit.

Can be **called/invoked** as:

- ❑ A function (previous examples).
- ❑ A method.
- ❑ A constructor.

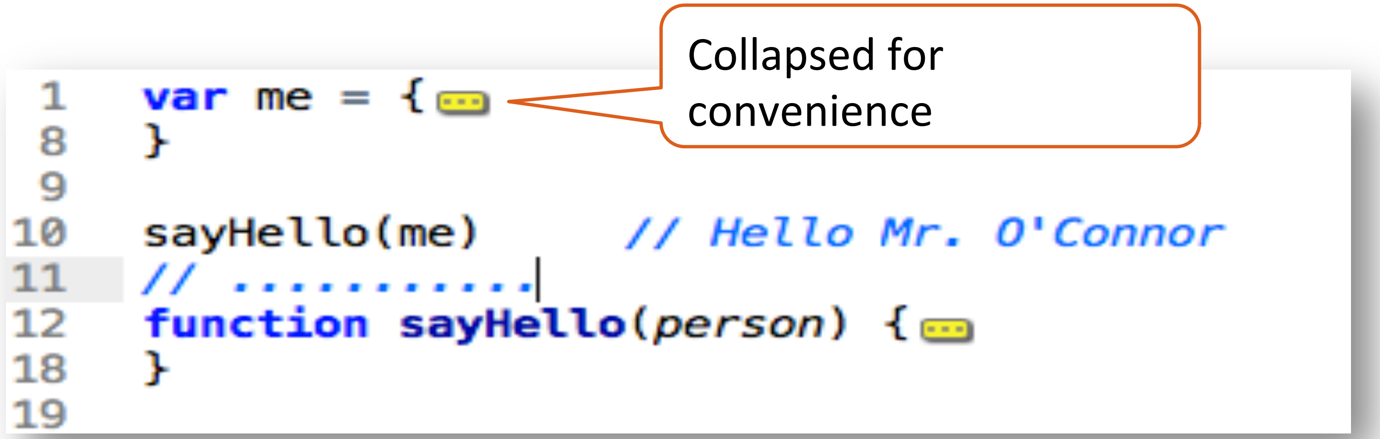
Function Declarations

Define a function using the syntax:

function name(...) { ... }

Function definitions are “hoisted” to the top of the current scope.

- You can use a function before it is defined – like function-scoped variables.



```
var me = {
}

sayHello(me)        // Hello Mr. O'Connor
// ...........
function sayHello(person) {
}

```

Can also define functions inside other functions – same scoping rules as variables.

Function Expressions

Defines a function using the syntax:

```
var name = function( ... ) { ... }
```

Unlike function declarations, there is no “hoisting”.

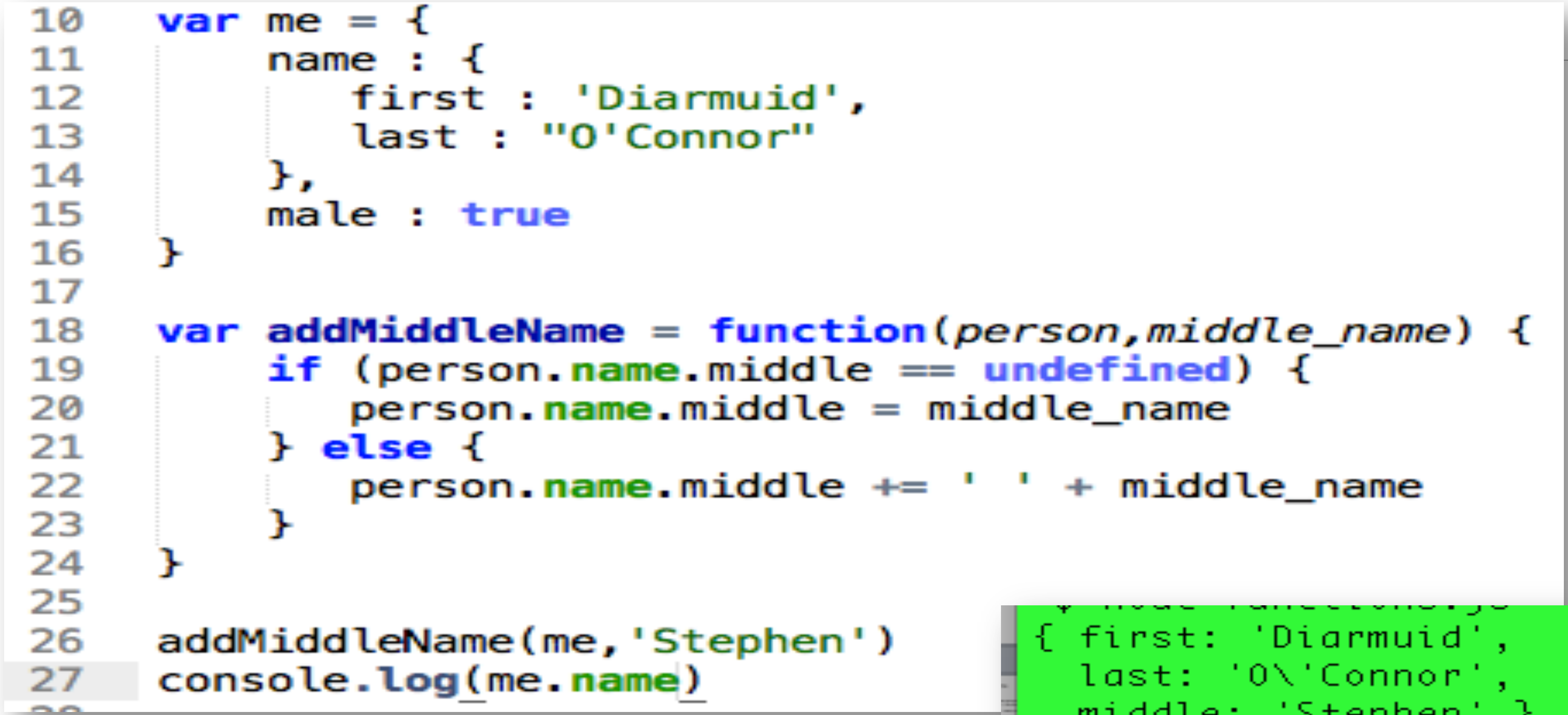
- You can’t use the function before it is defined, because the variable referencing the function has no value, yet.

Useful for dynamically created functions.

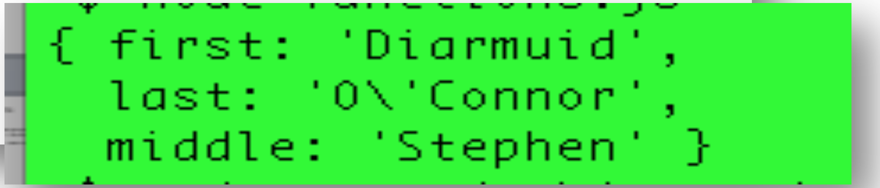
Called in the same way as function declarations:

```
name( argument1, argument2, ... )
```

Function Expressions



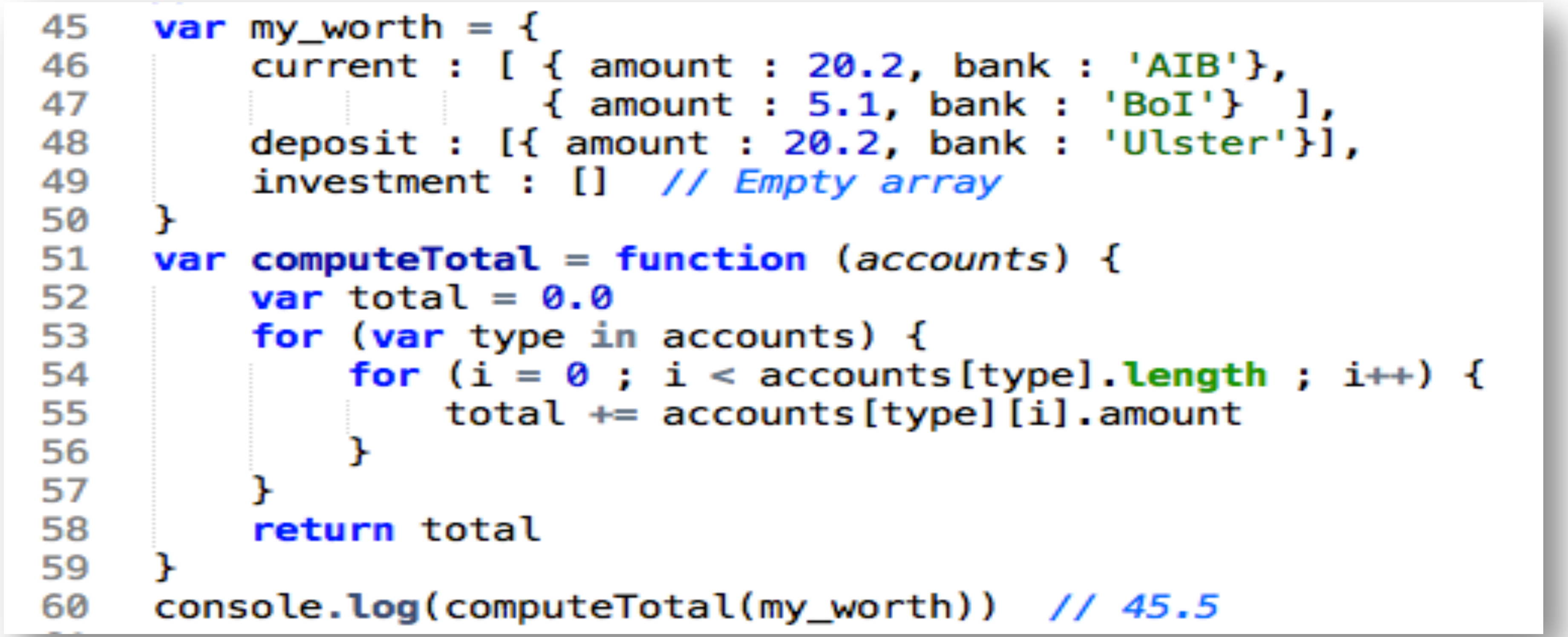
```
var me = {
    name : {
        first : 'Diarmuid',
        last : "O'Connor"
    },
    male : true
}

var addMiddleName = function(person,middle_name) {
    if (person.name.middle == undefined) {
        person.name.middle = middle_name
    } else {
        person.name.middle += ' ' + middle_name
    }
}

addMiddleName(me,'Stephen')
console.log(me.name)
```



```
{ first: 'Diarmuid',
  last: 'O\'Connor',
  middle: 'Stephen' }
```

Function Returns

Typically, functions perform some logic AND return a result.



```
var my_worth = {
    current : [ { amount : 20.2, bank : 'AIB'},
                { amount : 5.1, bank : 'BoI'}  ],
    deposit : [{ amount : 20.2, bank : 'Ulster'}],
    investment : []  // Empty array
}
var computeTotal = function (accounts) {
    var total = 0.0
    for (var type in accounts) {
        for (i = 0 ; i < accounts[type].length ; i++) {
            total += accounts[type][i].amount
        }
    }
    return total
}
console.log(computeTotal(my_worth))  // 45.5
```

[A function without a return statement will return ‘undefined’]

Methods

A property value of an object can be a function, termed a method.

The same form of function definition as function expressions.

Syntax: var obj = { …….
                    methodX : function(….) { …. },
                   …….. }

Calling method syntax: obj.methodX(….)

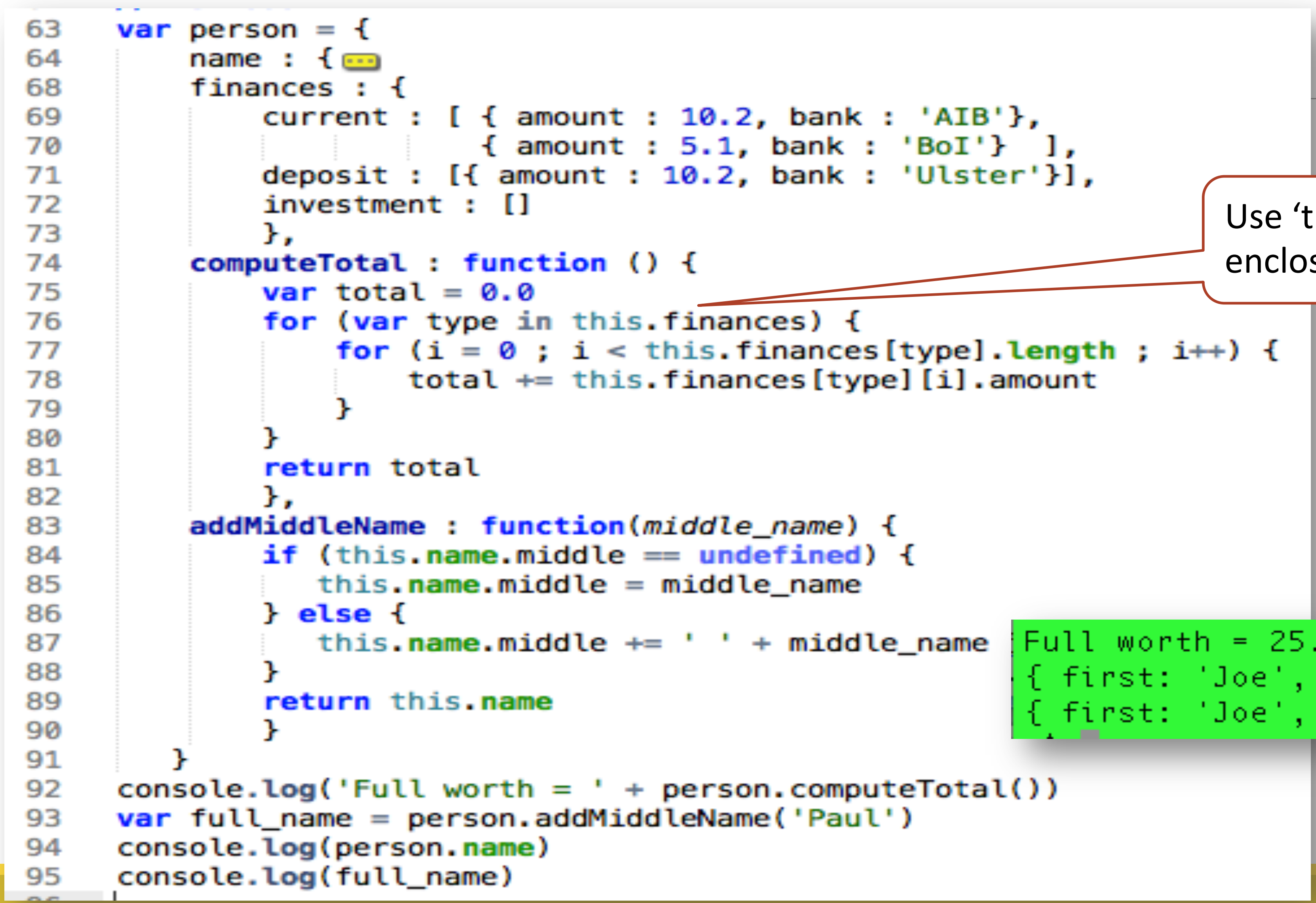
Methods of an object can be redefined or added at any time.

- JS is dynamic!!

Methods must be defined before use.

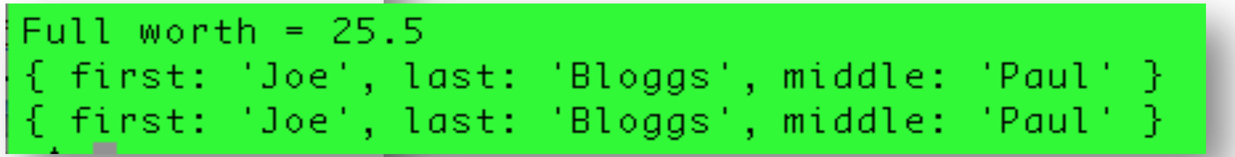
[*A bit on Application design* – The dominant design methodology encourages encapsulating state (data) and behavior (methods) into units called classes. In JS, most custom objects are a mix of state and methods, where the latter manipulates the state. **]**

Methods



```
var person = {
    name : {
    finances : {
        current : [ { amount : 10.2, bank : 'AIB'},
                    { amount : 5.1, bank : 'BoI'}  ],
        deposit : [{ amount : 10.2, bank : 'Ulster'}],
        investment : []
        },
    computeTotal : function () {
        var total = 0.0
        for (var type in this.finances) {
            for (i = 0 ; i < this.finances[type].length ; i++) {
                total += this.finances[type][i].amount
            }
        }
        return total
        },
    addMiddleName : function(middle_name) {
        if (this.name.middle == undefined) {
           this.name.middle = middle_name
        } else {
           this.name.middle += ' ' + middle_name
        }
        return this.name
        }
    }
console.log('Full worth = ' + person.computeTotal())
var full_name = person.addMiddleName('Paul')
console.log(person.name)
console.log(full_name)
```

Use 'this' to reference the enclosing object



```
Full worth = 25.5
{ first: 'Joe', last: 'Bloggs', middle: 'Paul' }
{ first: 'Joe', last: 'Bloggs', middle: 'Paul' }
```

Methods

Syntax comparison:

- Function:

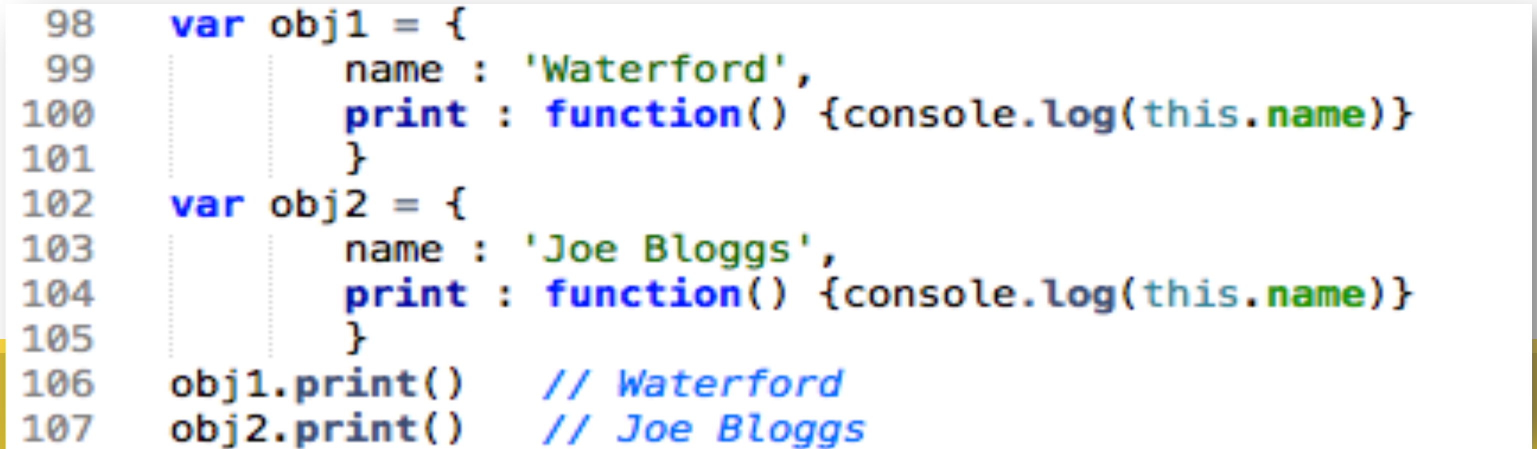
  computeTotal(person) addMiddleName(person,’ Paul')

- Method:

  person.computeTotal() person.addMiddleName(me,’ Paul’)

The special ‘this’ variable.

- Always references the enclosing object.
- Used by methods to access properties of the enclosing object.



```
var obj1 = {
        name : 'Waterford',
        print : function() {console.log(this.name)}
        }
var obj2 = {
        name : 'Joe Bloggs',
        print : function() {console.log(this.name)}
        }
obj1.print()   // Waterford
obj2.print()   // Joe Bloggs
```

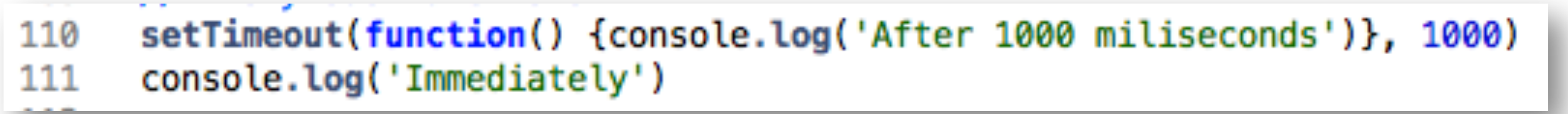
Anonymous functions

You can define a function without giving it a name:

function(...) { …. }

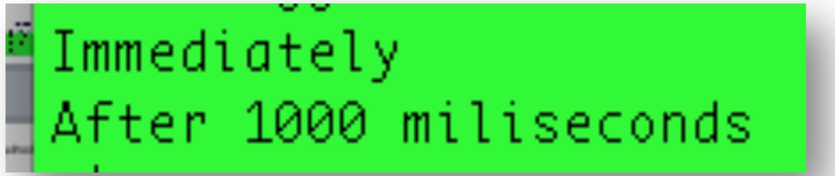
Mainly used for "callbacks" - when a function/method needs another function as an argument, which it calls.

- Example - The setTimeout system function.



```
setTimeout(function() {console.log('After 1000 miliseconds')}, 1000)
console.log('Immediately')
```

[Note: Any type of function (declaration, expression, method) can be used as a callback, not just anonymous functions.]

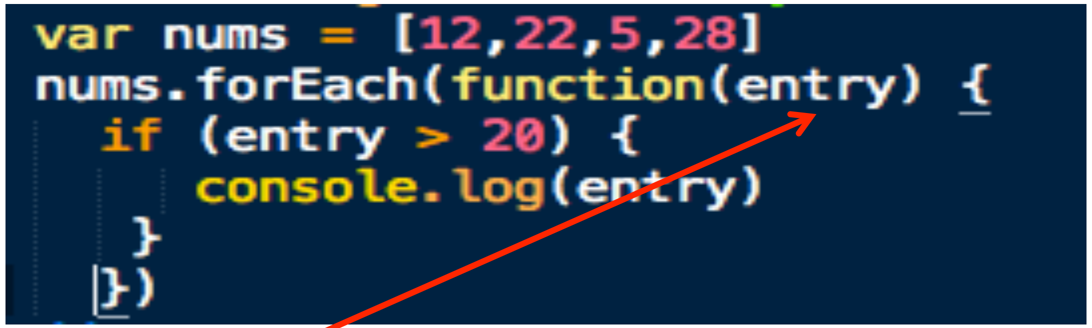


```
Immediately
After 1000 miliseconds
```

Anonymous functions

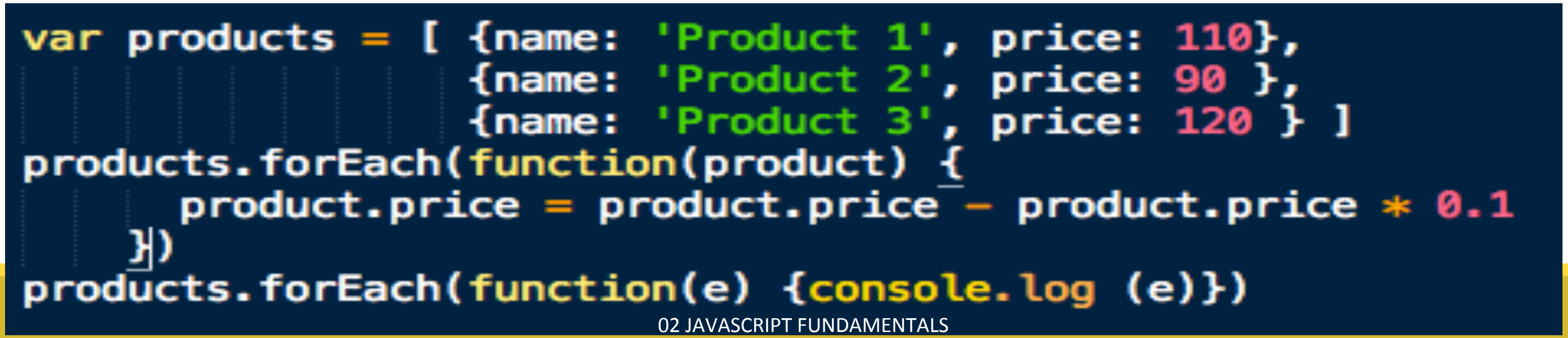
A more elegant way of processing an array.

- Objective: Display every number > 20 from the array.



```
var nums = [12,22,5,28]
nums.forEach(function(entry) {
  if (entry > 20) {
     console.log(entry)
  }
})
```

- The anonymous function is called by forEach(), once for each entry in the array. The function's parameter (entry) will be set to the current array entry being processed.



```
var products = [ {name: 'Product 1', price: 110},
                 {name: 'Product 2', price: 90 },
                 {name: 'Product 3', price: 120 } ]
products.forEach(function(product) {
      product.price = product.price - product.price * 0.1
  })
products.forEach(function(e) {console.log (e)})
```

Constructors

The object literal syntax is not efficient for creating multiple objects of a common ‘type’.

- Efficiency = Amount of source code.

```
var customer1 = { name ’ Joe Bloggs’ ,
        address  :  ‘1 Main St’ ,
        finances : {. . . . . },
        computeTotal : function () { . . . . },
        adjustFinance : function (change) { . . . }
   }
var customer2 = { name ’ Pat Smith’ ,
        address  :  ‘2 High St’ ,
        finances : {. . . . . },
        computeTotal : function () { . . . . },
        adjustFinance : function (change) { . . . }
   }
var customer3 = . . . . .
```

Constructors solve this problem

Constructors.

Constructor - Function for creating (constructing) an object of *a custom type*.

- Custom type examples: Customer, Product, Order, Student, Module, Lecture.

- Idea borrowed from class-based languages, e.g. Java.
  - No classes in Javascript.

Convention: Capitalize function name to distinguish it from ordinary functions.

```
function Foo(. . . ) { ... }
```

Constructor call must be preceded by the new operator.

```
var a_foo = new Foo( . . . )
```

Constructors

What happens when a constructor is called?

1. A new (empty) object is created, ie. { } .
2. The ***this*** variable is set to the new object.
3. The function is executed.
4. The default return value is the object referenced by this.

```
function Customer (name_in,address_in,finances_in) {
        this.name = name_in
        this.address = address_in
        this.finances = finances_in
        this.computeTotal = function () { . . . . }
        this.changeFinannce = function (change) { . . . . }
}
var customer1 = new Customer ('Joe Bloggs','I Main St.', {. . . } )
var customer1 = new Customer ('Pat Smith','2 High St.', {. . . } )
console.log(customer1.name)     // Joe Bloggs
var total = customer1.computeTotal()
```

JavaScript

Questions?