Let's Have a Cup of CoffeeScript

Nicolás Sanguinetti

http://github.com/foca

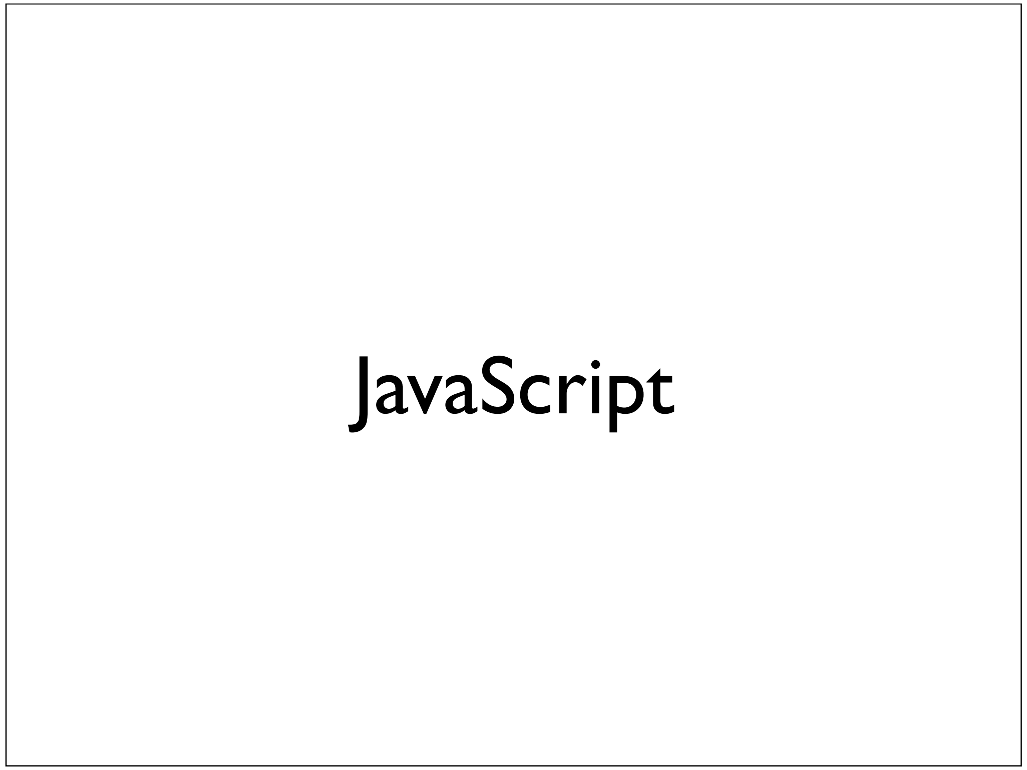
@godfoca

cubox

Thank organisers
Thank cubox

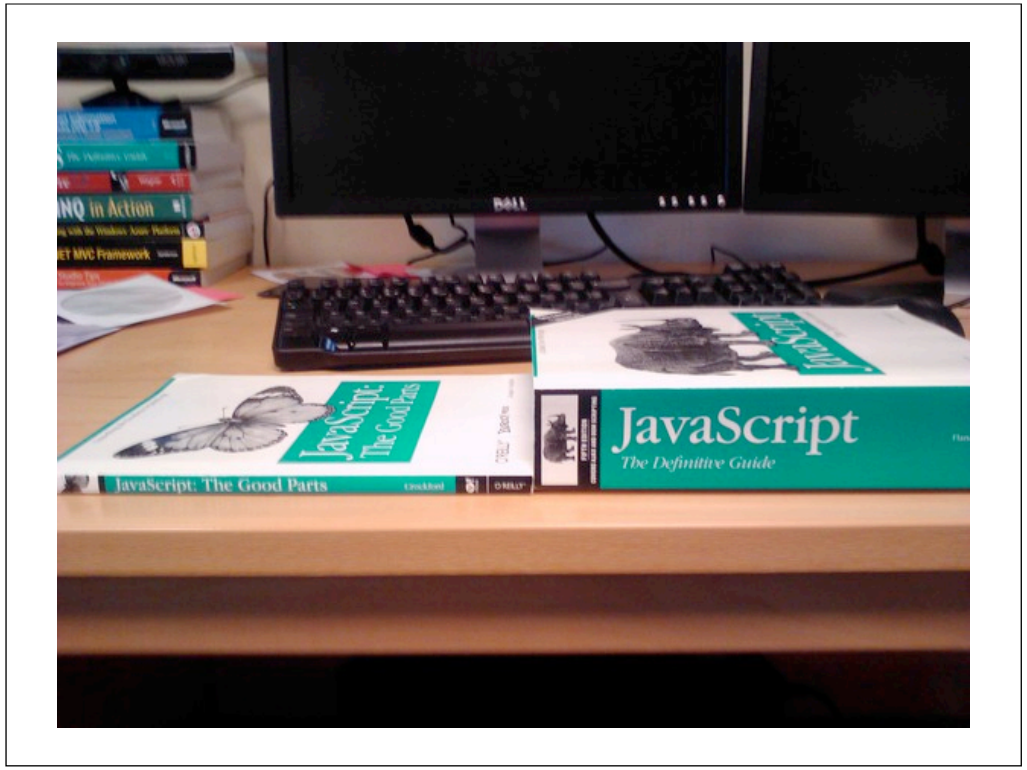
CoffeeScript

Here to talk CoffeeScript
But first: JavaScript



JavaScript

Quick Hack: 10 days
Lisp and self, political: C syntax
But, web’s lingua franca
Some Good Parts
Some Not-So-Good Parts



Not-So-Good outweigh Good
Coffee focuses on Good
Hides bad.

But is it worth it?

that’s what we’re here to find out
large community
big names using it like 37s

An Example

An Example

```
var square = function(num) {
  return num * num;
}
var list = [1, 2, 3, 4, 5];
var squares = [];
for (var i = 0; i < list.length; i++) {
  squares.push(square(list[i]));
}
```

An Example

```
var square = function(num) {
  return num * num;
}
var list = [1, 2, 3, 4, 5];
var squares = [];
for (var i = 0; i < list.length; i++) {
  squares.push(square(list[i]));
}
```

An Example

```
var square = function(num) {
  return num * num;
}
var list = [1, 2, 3, 4, 5];
var squares = [];
for (var i = 0; i < list.length; i++) {
  squares.push(square(list[i]));
}
```

An Example

```
var square = function(num) {
  return num * num;
}
var list = [1, 2, 3, 4, 5];
var squares = [];
for (var i = 0; i < list.length; i++) {
  squares.push(square(list[i]));
}
```

An Example

```
var square = function(num) {
  return num * num;
}
var list = [1, 2, 3, 4, 5];
var squares = [];
for (var i = 0; i < list.length; i++) {
  squares.push(square(list[i]));
}
```

An Example

```
square = (num) -> num * num
list = [1..5]
squares = (square n for n in list)
```

An Example

```
square = (num) -> num * num
list = [1..5]
squares = (square n for n in list)
```

An Example

```
square = (num) -> num * num
list = [1..5]
squares = (square n for n in list)
```

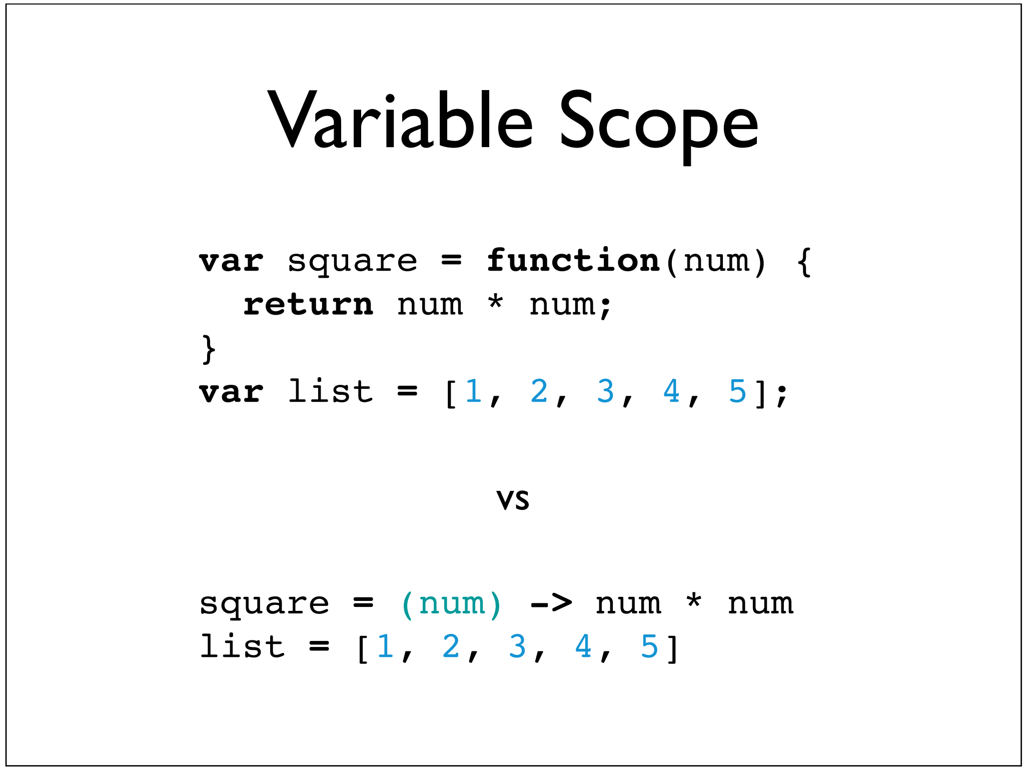
An Example

```
square = (num) -> num * num
list = [1..5]
squares = (square n for n in list)
```

An Example

```
square = (num) -> num * num
list = [1..5]
squares = (square n for n in list)
```

Variable Scope



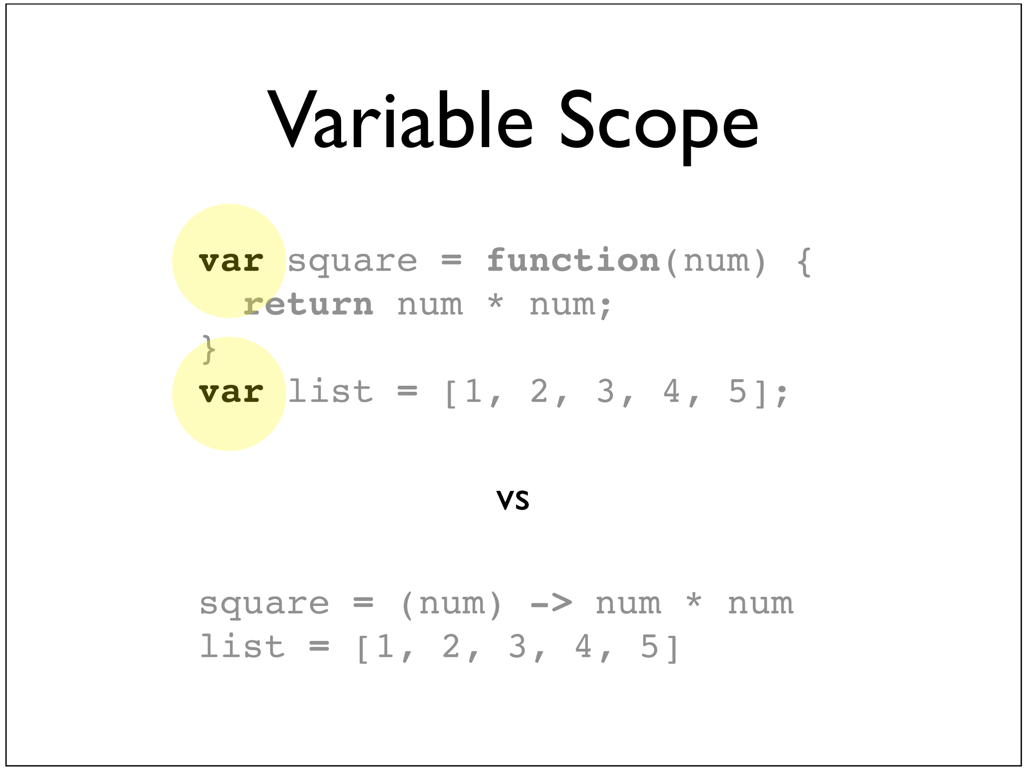
Variable Scope

```
var square = function(num) {
  return num * num;
}
var list = [1, 2, 3, 4, 5];
```

vs

```
square = (num) -> num * num
list = [1, 2, 3, 4, 5]
```

first thing CS: var scope



Variable Scope

```
var square = function(num) {
  return num * num;
}
var list = [1, 2, 3, 4, 5];
```

vs

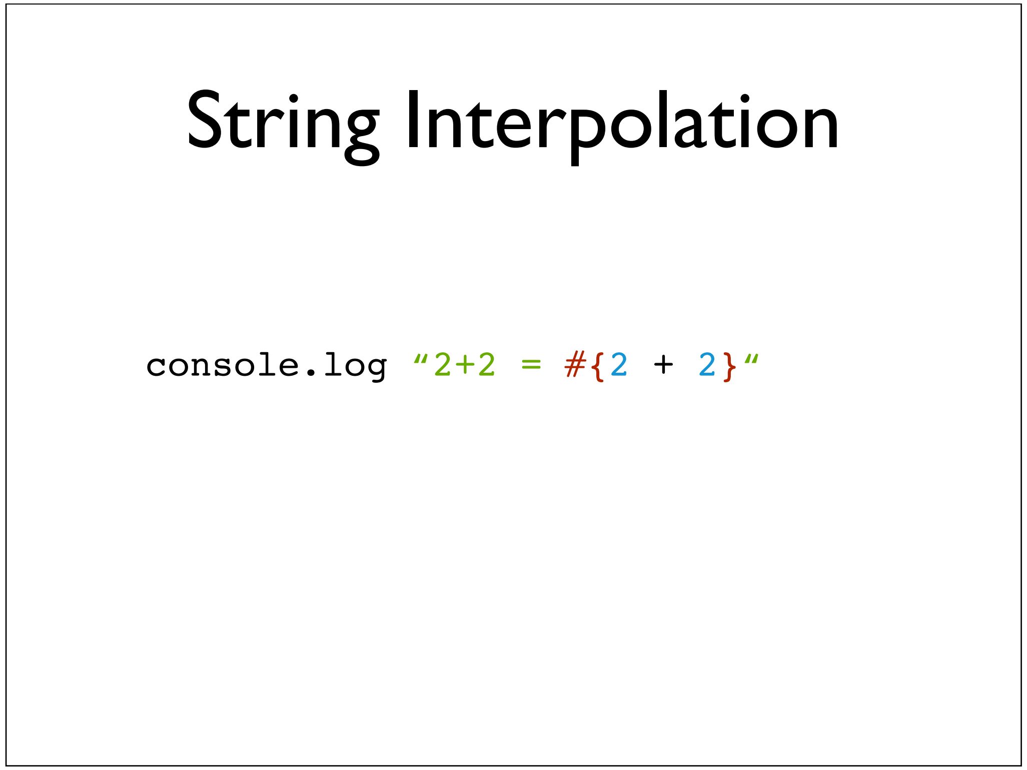
```
square = (num) -> num * num
list = [1, 2, 3, 4, 5]
```

no “var”
very common mistake: global
CS makes everything local to scope

String Interpolation

String Interpolation

```
name = “India“
console.log “Hi #{name}“
```

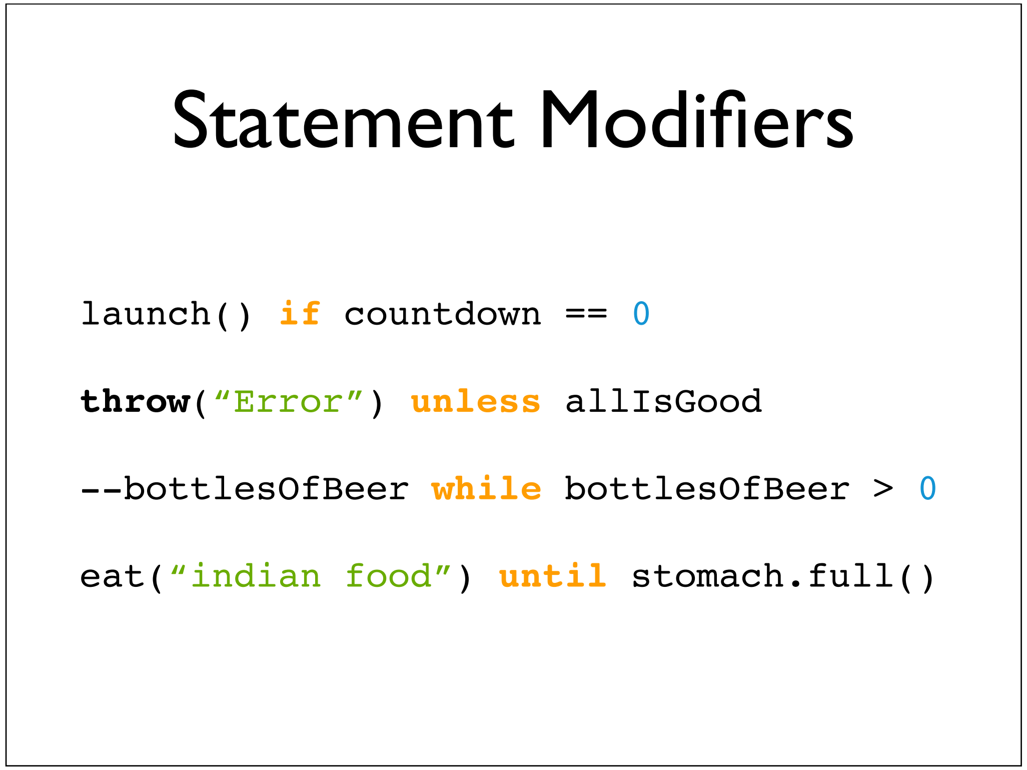


String Interpolation

```
console.log "2+2 = #{2 + 2}"
```

any expression can be interpolated

Statement Modifiers



Statement Modifiers

```
launch() if countdown == 0

throw("Error") unless allIsGood

--bottlesOfBeer while bottlesOfBeer > 0

eat("indian food") until stomach.full()
```

same as in ruby
postfix conditions

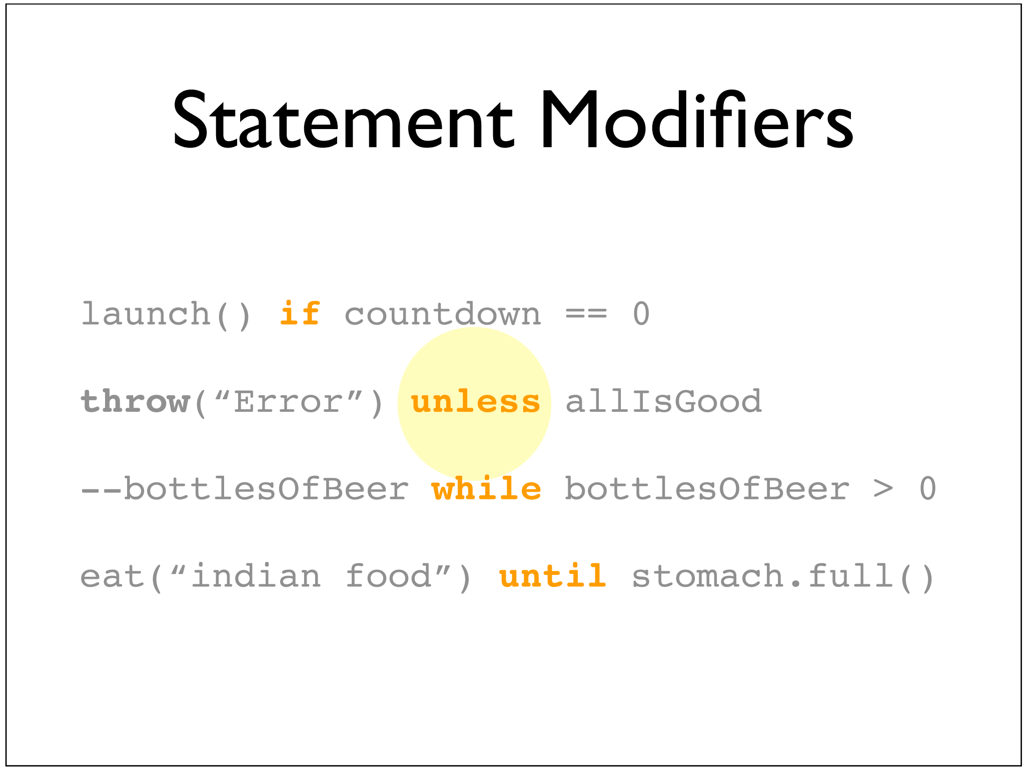
Statement Modifiers

```
launch() if countdown == 0

throw(“Error”) unless allIsGood

--bottlesOfBeer while bottlesOfBeer > 0

eat(“indian food”) until stomach.full()
```



Statement Modifiers

```
launch() if countdown == 0

throw(“Error”) unless allIsGood

--bottlesOfBeer while bottlesOfBeer > 0

eat(“indian food”) until stomach.full()
```

if not

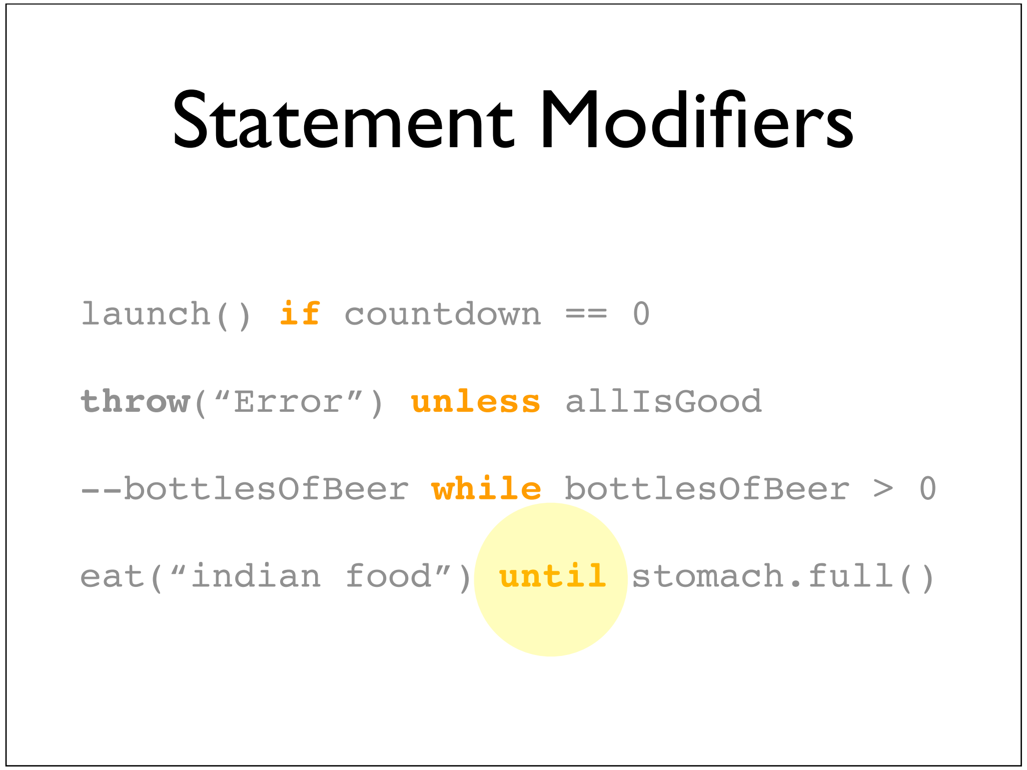
Statement Modifiers

```
launch() if countdown == 0

throw("Error") unless allIsGood

--bottlesOfBeer while bottlesOfBeer > 0

eat("indian food") until stomach.full()
```



Statement Modifiers

```
launch() if countdown == 0

throw("Error") unless allIsGood

--bottlesOfBeer while bottlesOfBeer > 0

eat("indian food") until stomach.full()
```

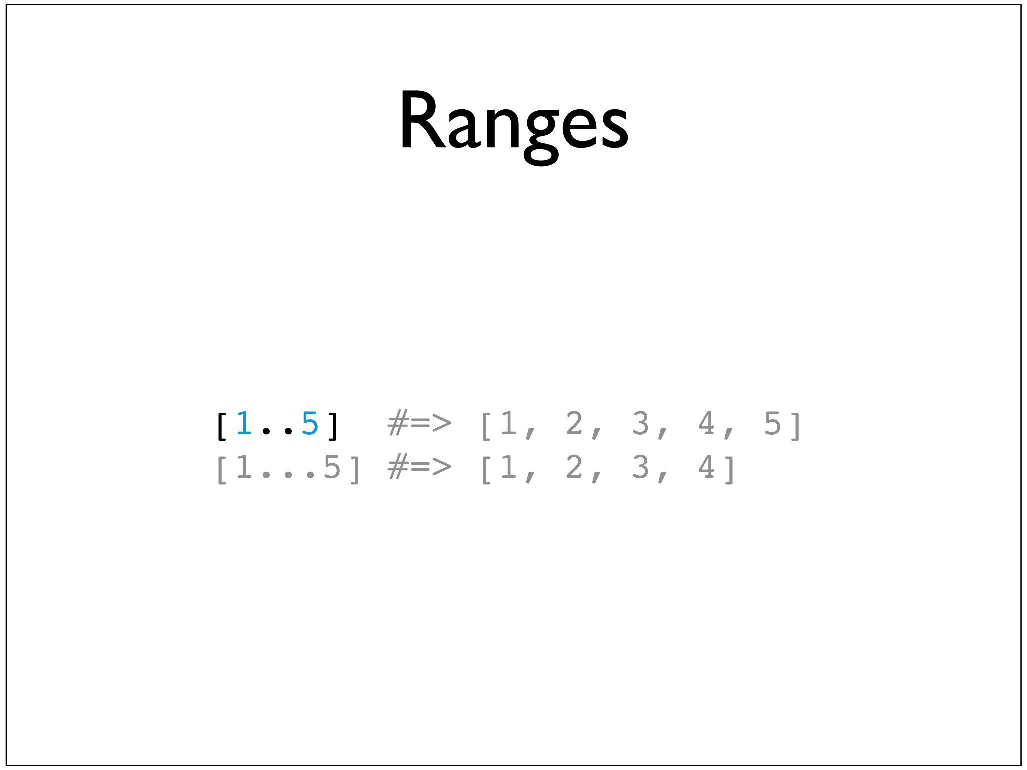
while not

Ranges

Ranges

```
[1..5]  #=> [1, 2, 3, 4, 5]
[1...5] #=> [1, 2, 3, 4]
```

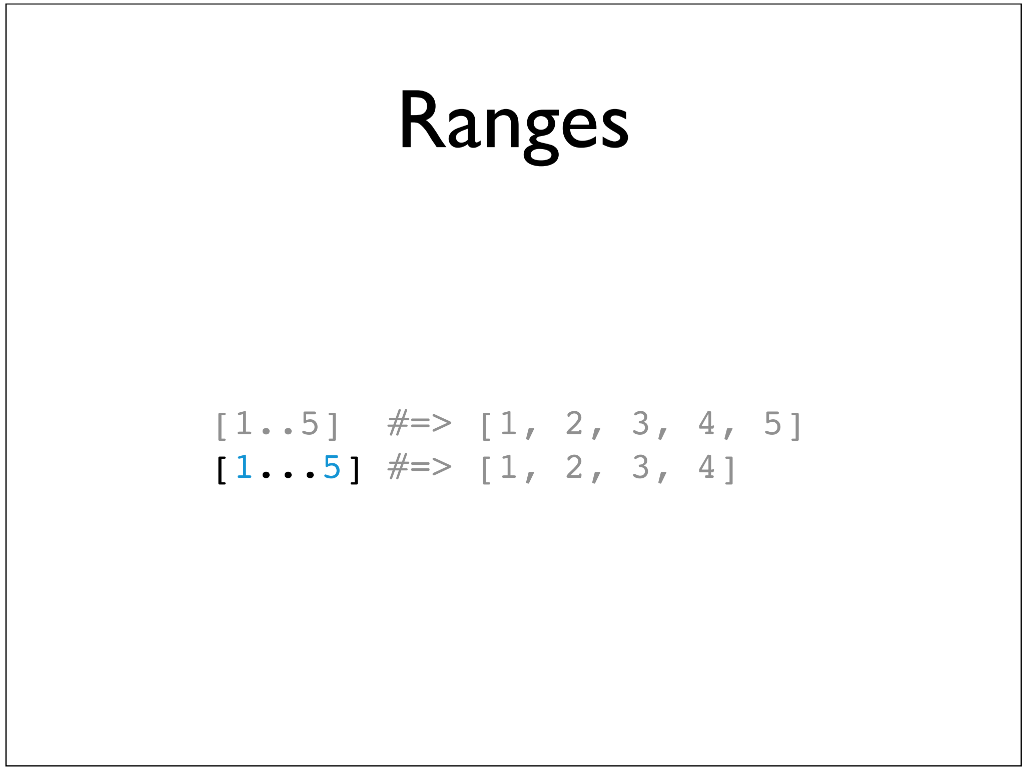
same as in ruby



Ranges

```
[1..5]  #=> [1, 2, 3, 4, 5]
[1...5] #=> [1, 2, 3, 4]
```

inclusive

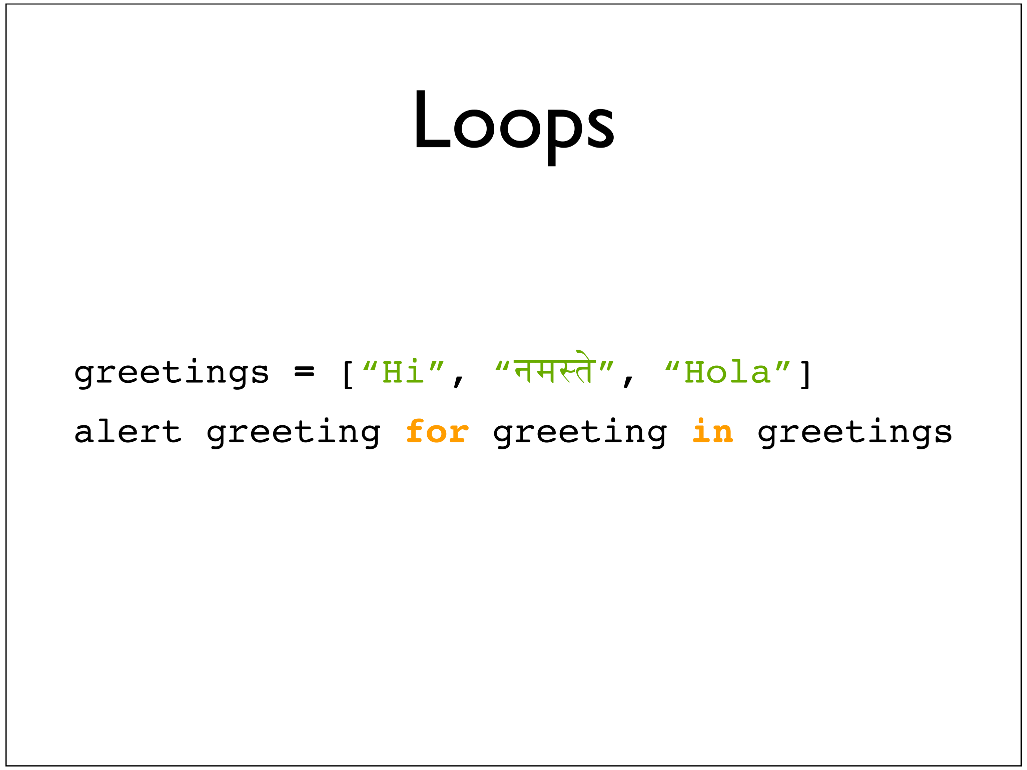


Ranges

```
[1..5]  #=> [1, 2, 3, 4, 5]
[1...5] #=> [1, 2, 3, 4]
```

exclusive

Loops



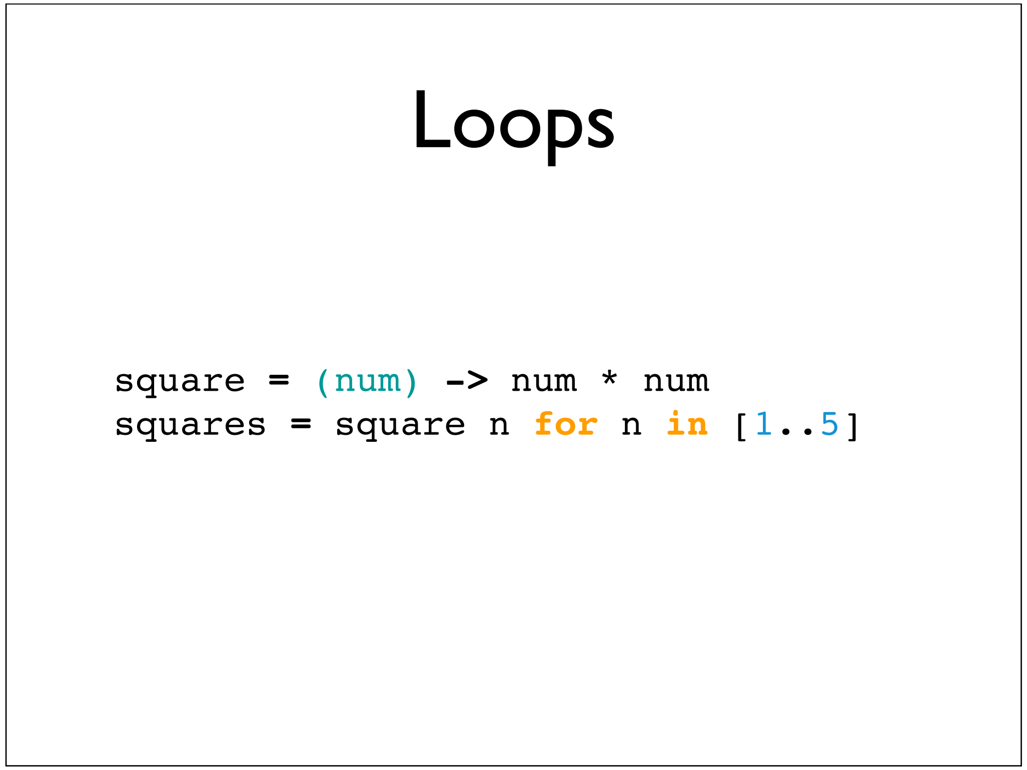
Loops

```
greetings = [“Hi”, “नमस्ते”, “Hola”]
alert greeting for greeting in greetings
```

iterate over a list

Loops

```
greetings = [“Hi”, “नमस्ते”, “Hola”]
alert greeting for greeting in greetings
```



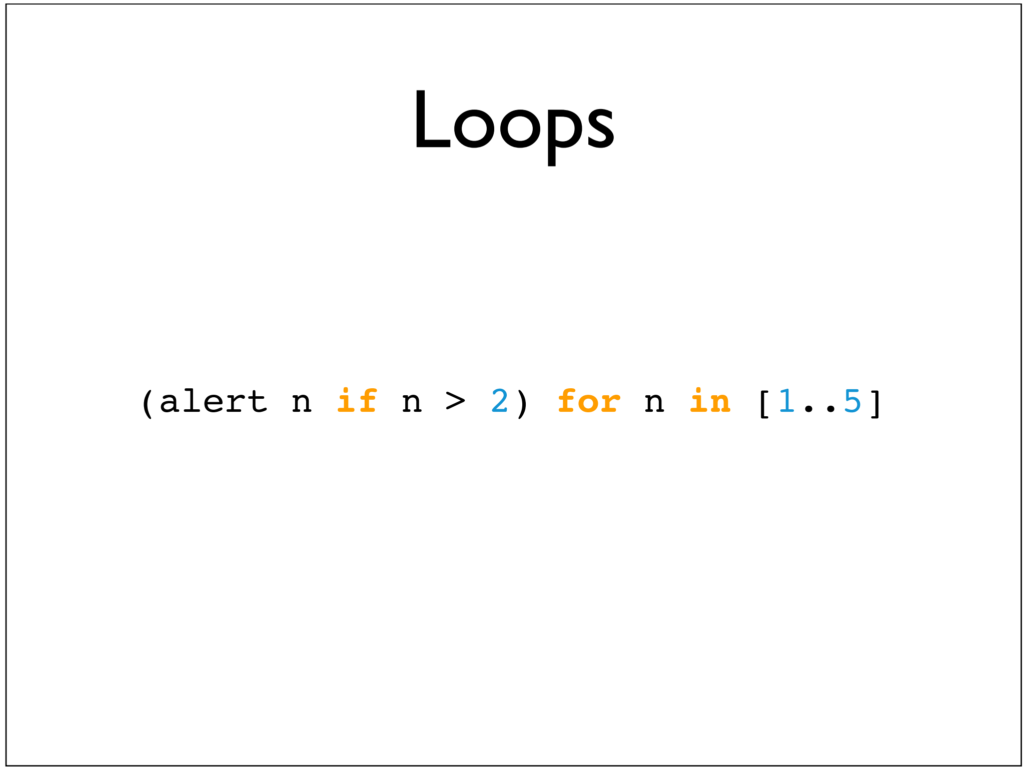
Loops

```
square = (num) -> num * num
squares = square n for n in [1..5]
```

also, map

Loops

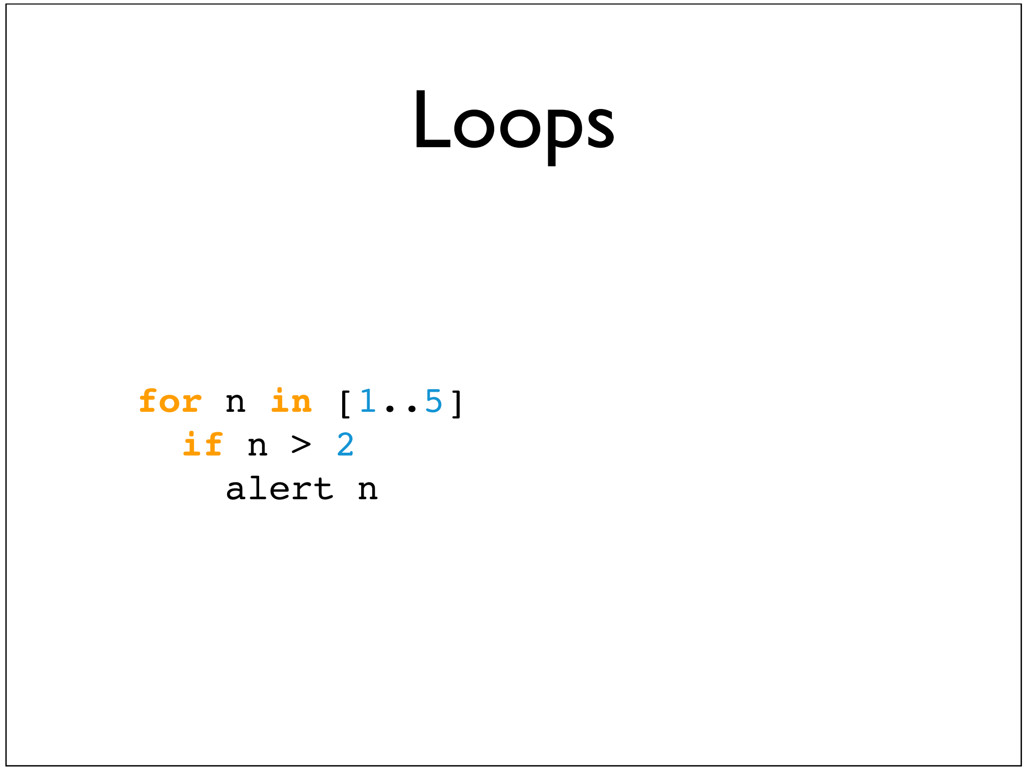
```
square = (num) -> num * num
squares = square n for n in [1..5]
```



Loops

```
(alert n if n > 2) for n in [1..5]
```

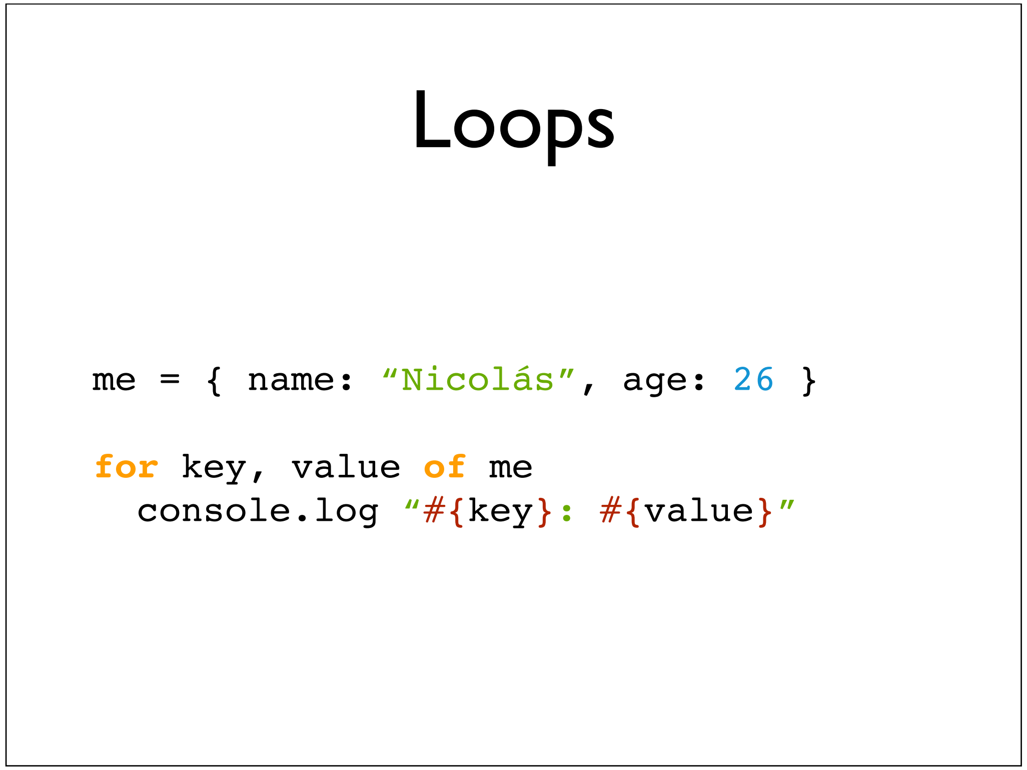
or select/filter
this can quickly get out of hand
better to use “traditional” for



Loops

```
for n in [1..5]
  if n > 2
    alert n
```

or select/filter
this can quickly get out of hand
better to use “traditional” for

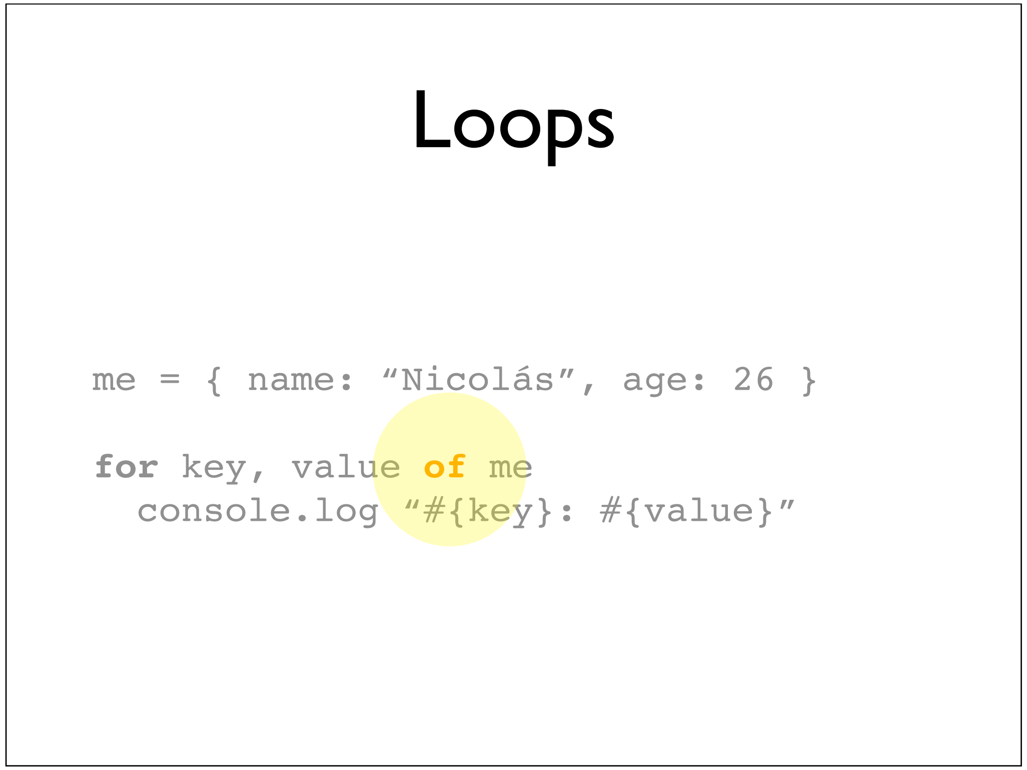


Loops

```
me = { name: “Nicolás”, age: 26 }

for key, value of me
  console.log “#{key}: #{value}”
```

can also iterate over key/val on objects

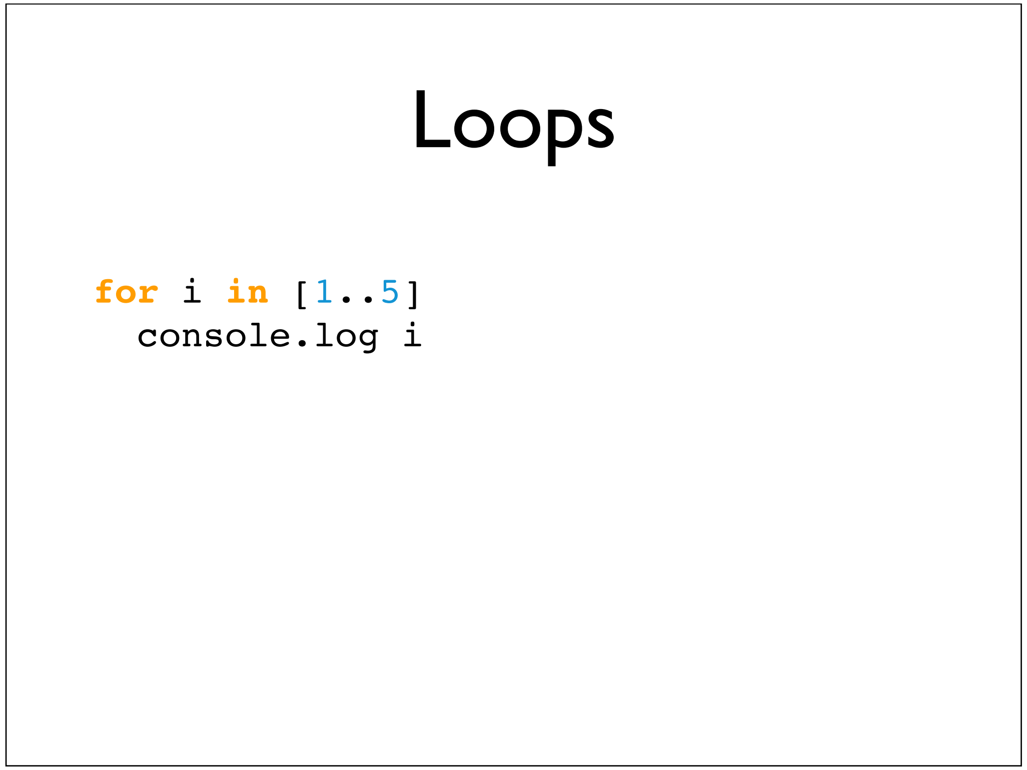


Loops

```
me = { name: “Nicolás”, age: 26 }

for key, value of me
  console.log “#{key}: #{value}”
```

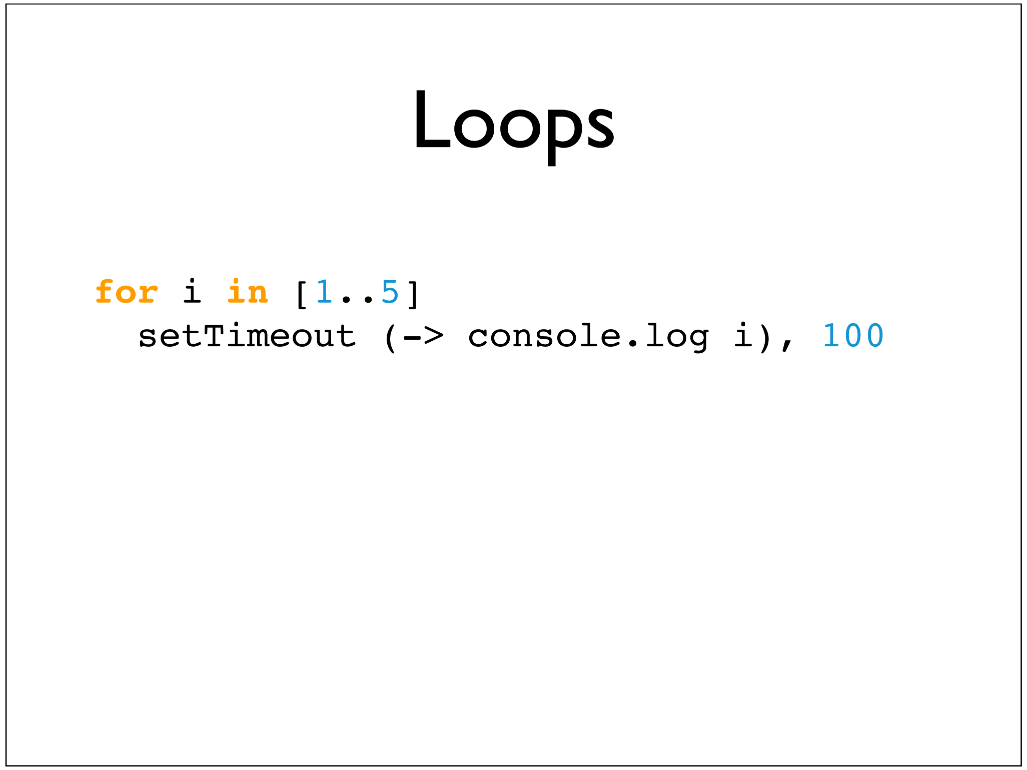
watch out!



Loops

```
for i in [1..5]
  console.log i
```

let’s take this simple for loop



Loops

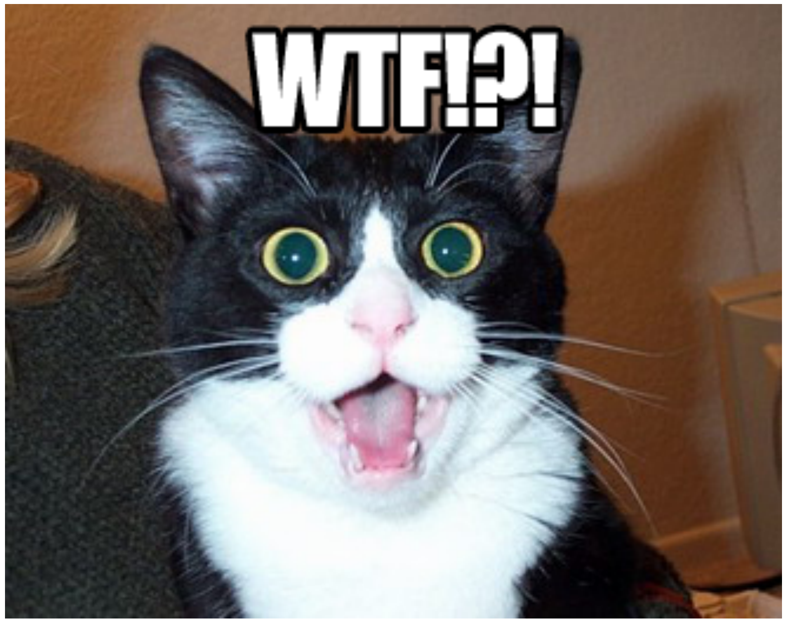
```
for i in [1..5]
  setTimeout (-> console.log i), 100
```

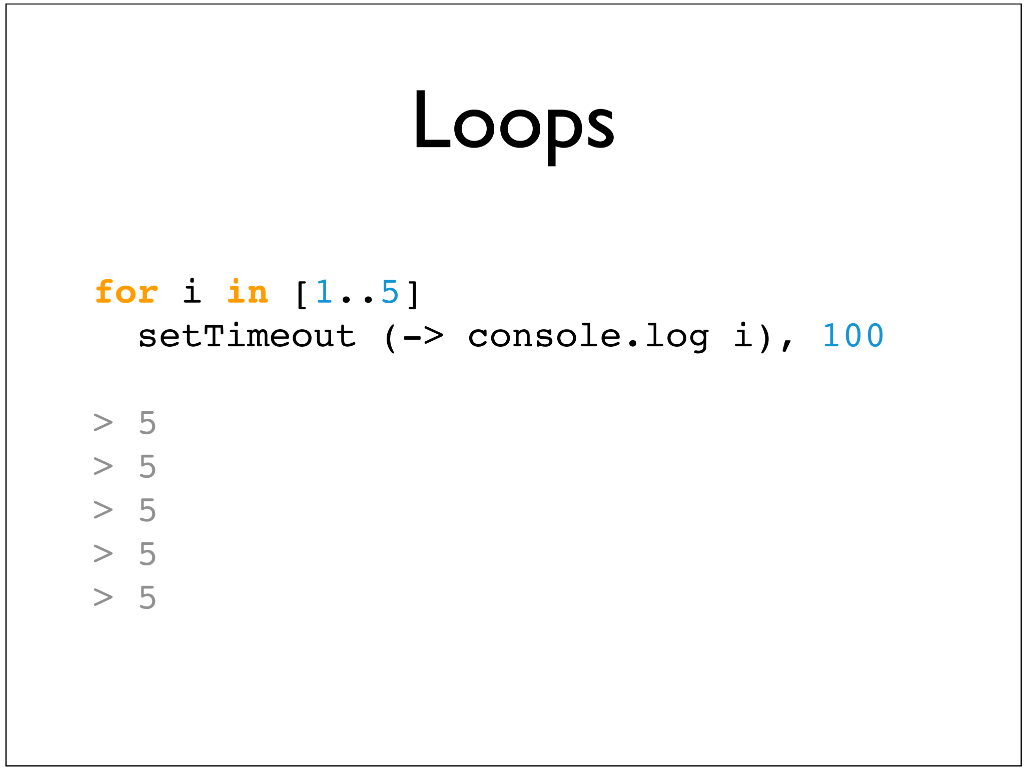
suppose we need timeouts
we wrap it in a function
but…

Loops

```
for i in [1..5]
  setTimeout (-> console.log i), 100
```

```
> 5
> 5
> 5
> 5
> 5
```


WTF!?!

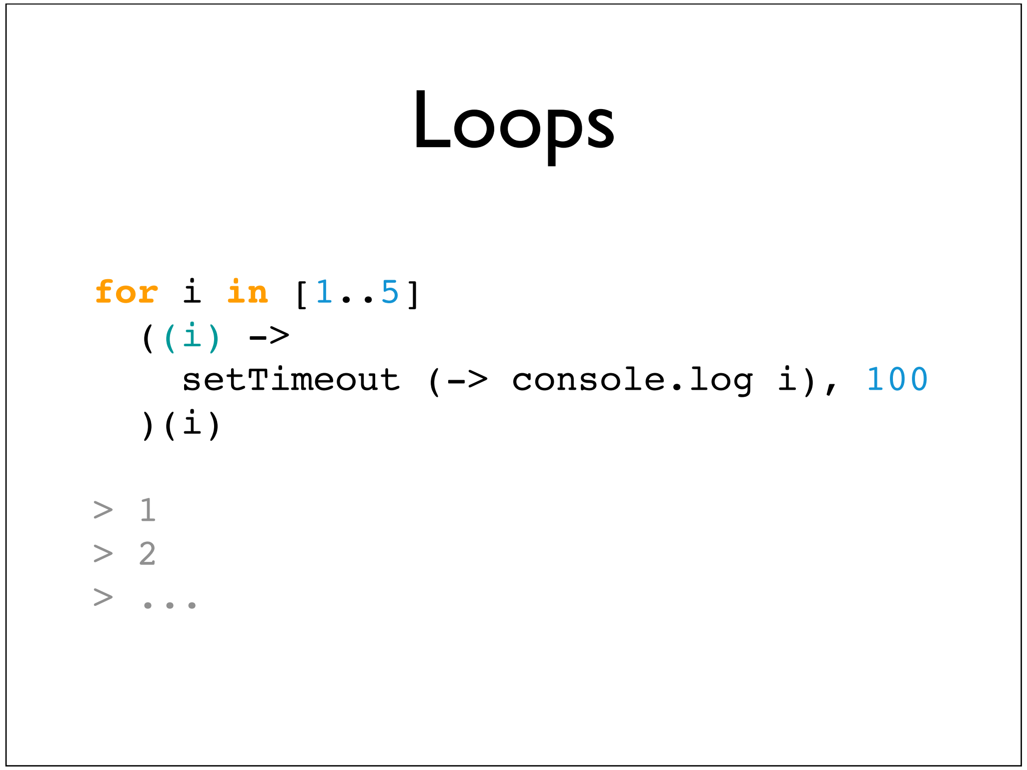


Loops

```
for i in [1..5]
  setTimeout (-> console.log i), 100

> 5
> 5
> 5
> 5
> 5
```

js passes last value to closure

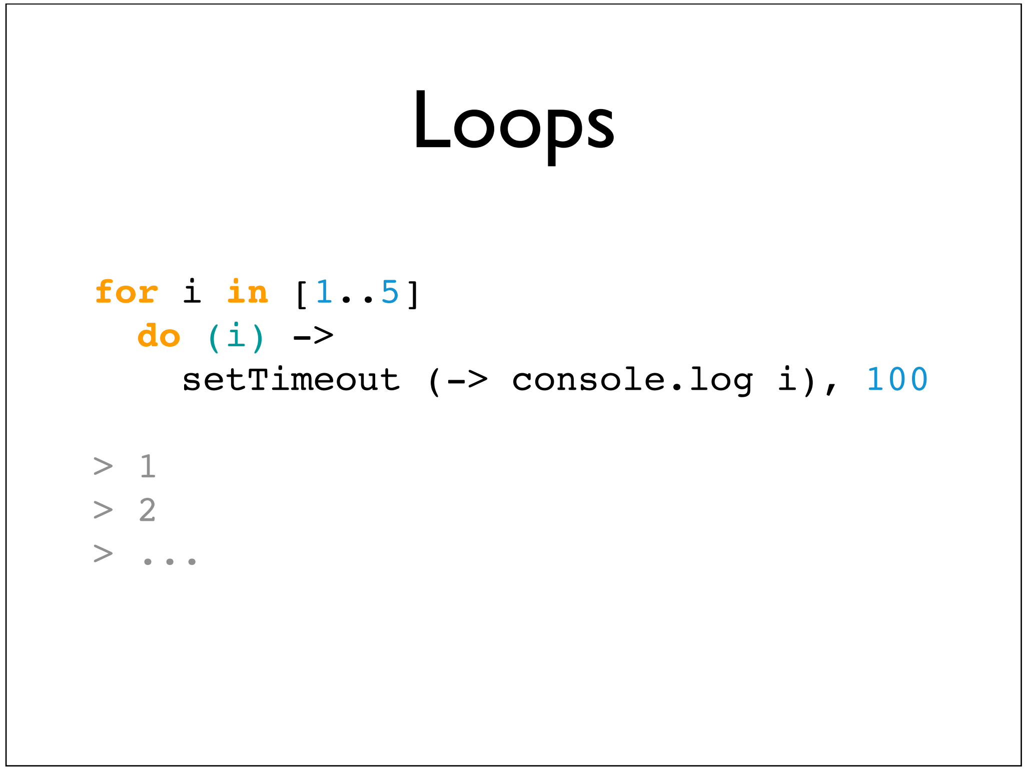


Loops

```
for i in [1..5]
  ((i) ->
    setTimeout (-> console.log i), 100
  )(i)

> 1
> 2
> ...
```

solution: autocalling fn
ensure closure gets proper val



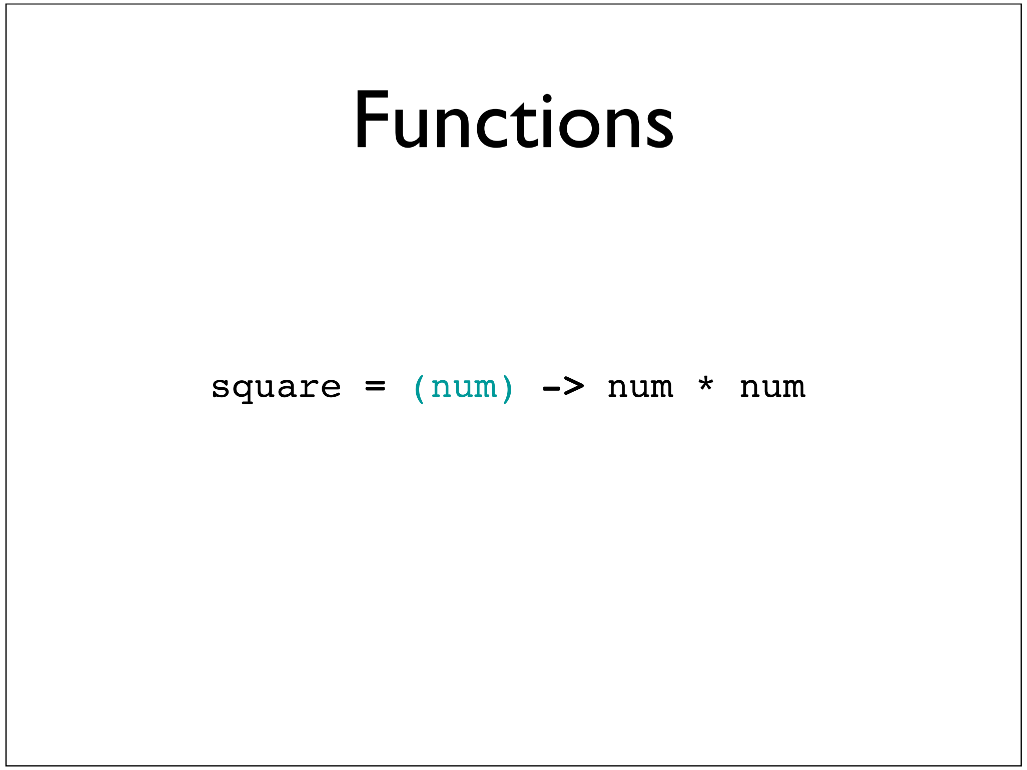
Loops

```
for i in [1..5]
  do (i) ->
    setTimeout (-> console.log i), 100

> 1
> 2
> ...
```

cs has a shorthand :)

Functions



Functions

```
square = (num) -> num * num
```

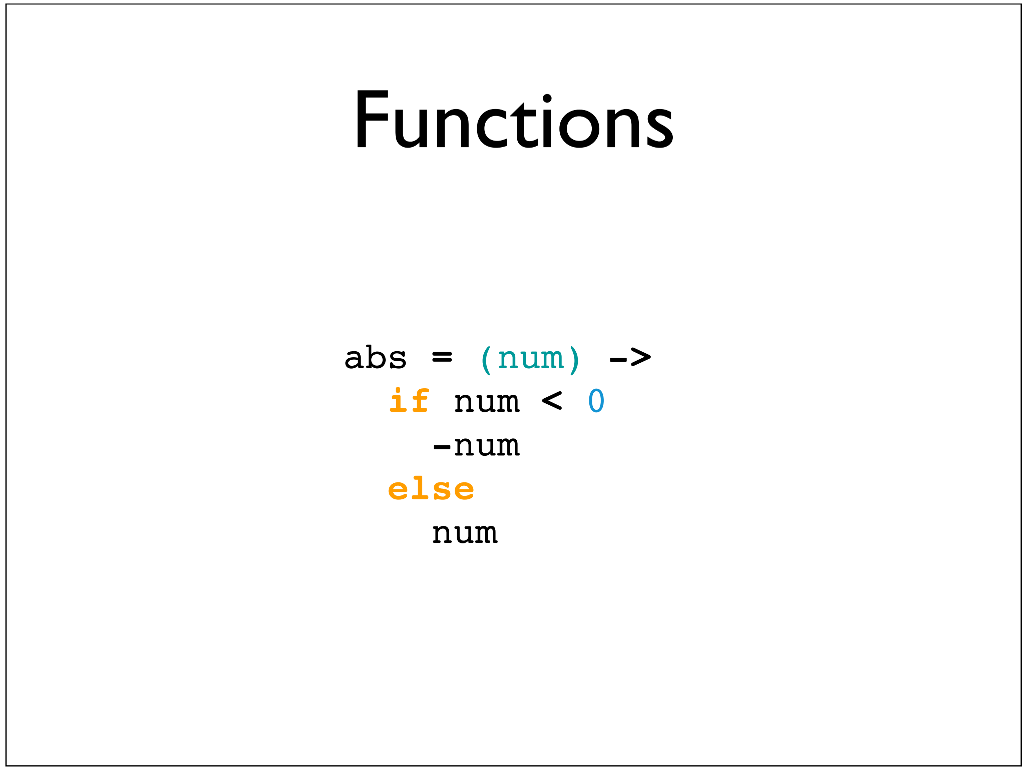
simple syntax
args, arrow, body
looks like ruby 1.9's blocks

Functions

```
square = (num) -> num * num
```

In Ruby

```
square = ->(num) { num * num }
```

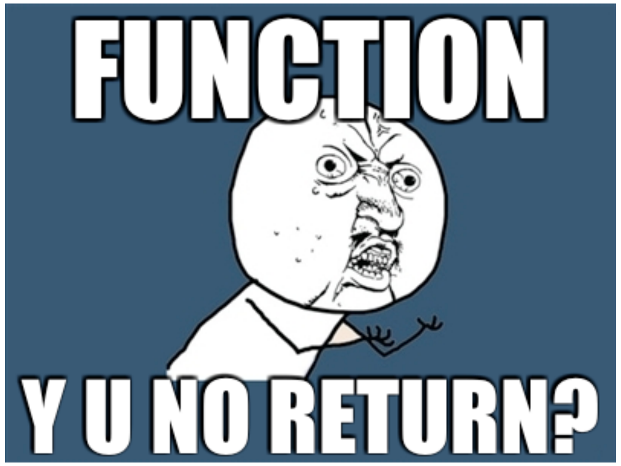


Functions

```
abs = (num) ->
  if num < 0
    -num
  else
    num
```

but something’s missing

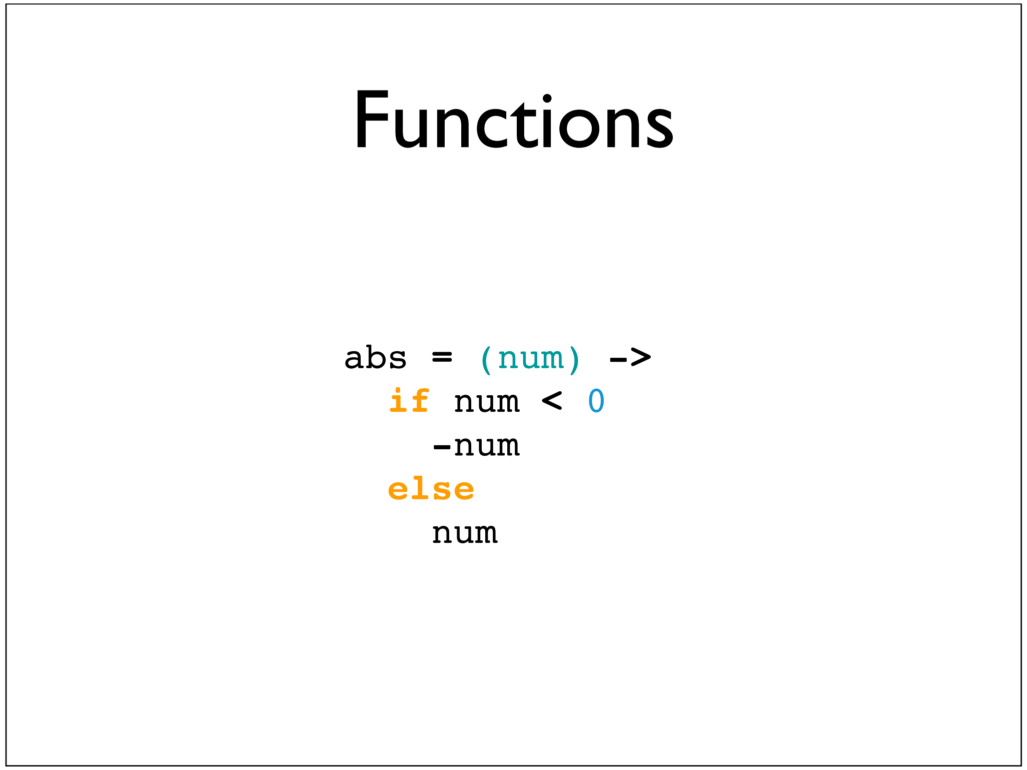
Functions



Functions

```
abs = (num) ->
  if num < 0
    return -num
  else
    return num
```

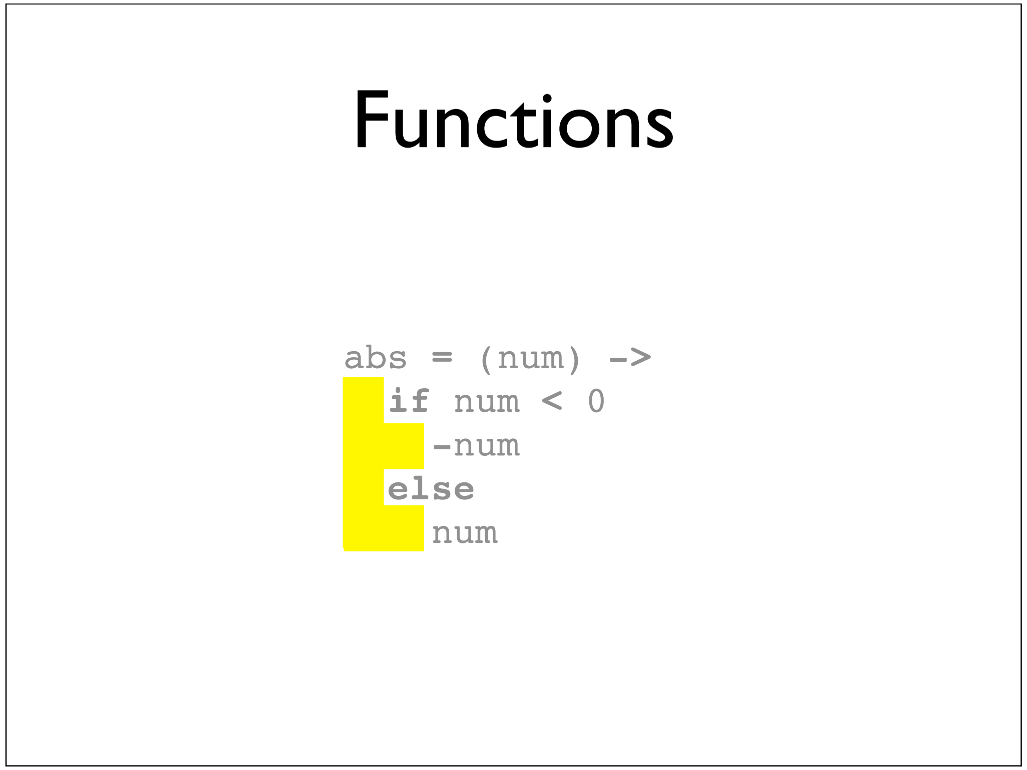
CS makes last statement return
even if it is “if”, “while”, etc
implicit return everywhere



Functions

```
abs = (num) ->
  if num < 0
    -num
  else
    num
```

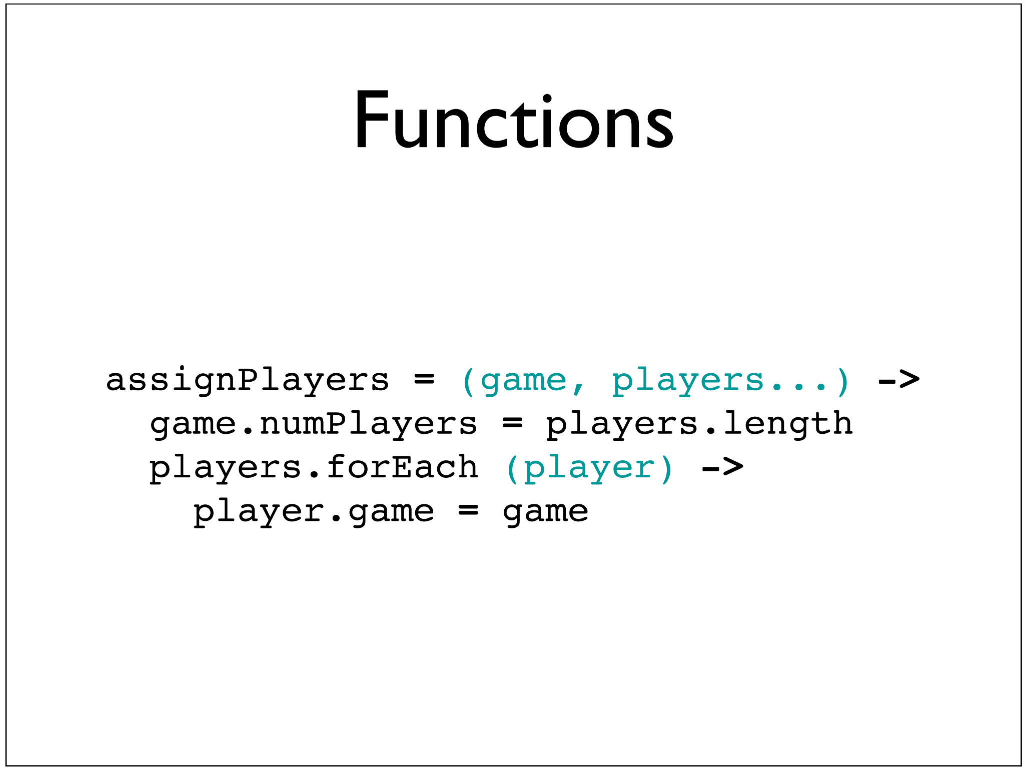
something else missing
no curly braces
no way to delimit function or if



Functions

```
abs = (num) ->
  if num < 0
    -num
  else
    num
```

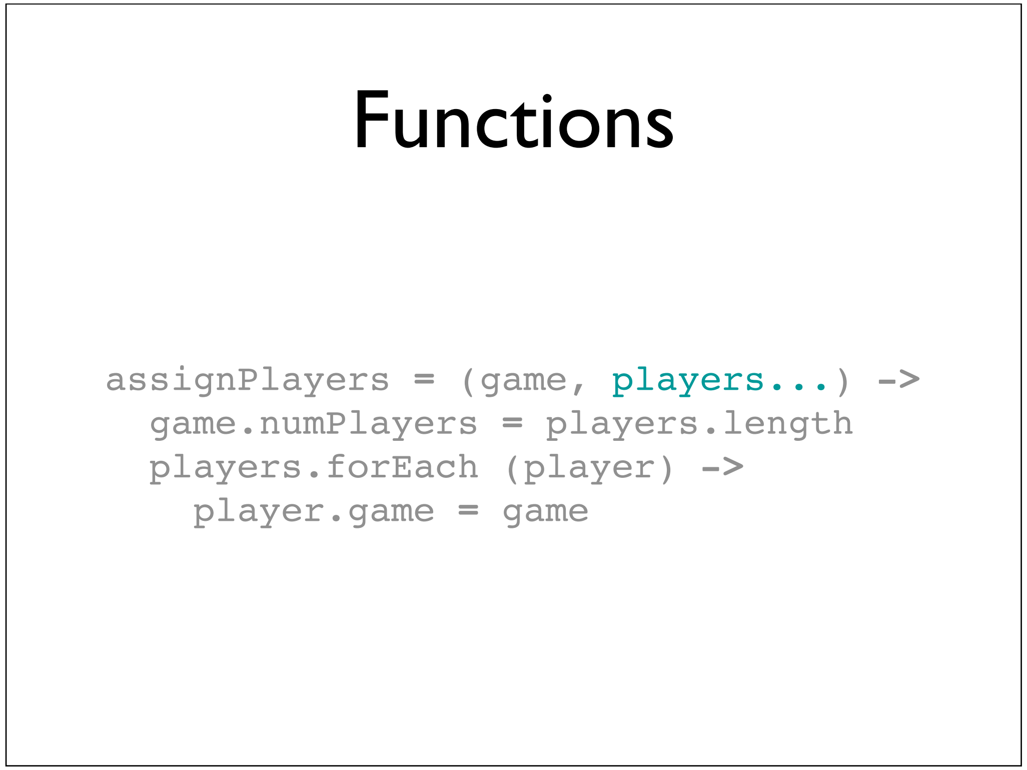
whitespace is significant



Functions

```
assignPlayers = (game, players...) ->
  game.numPlayers = players.length
  players.forEach (player) ->
    player.game = game
```

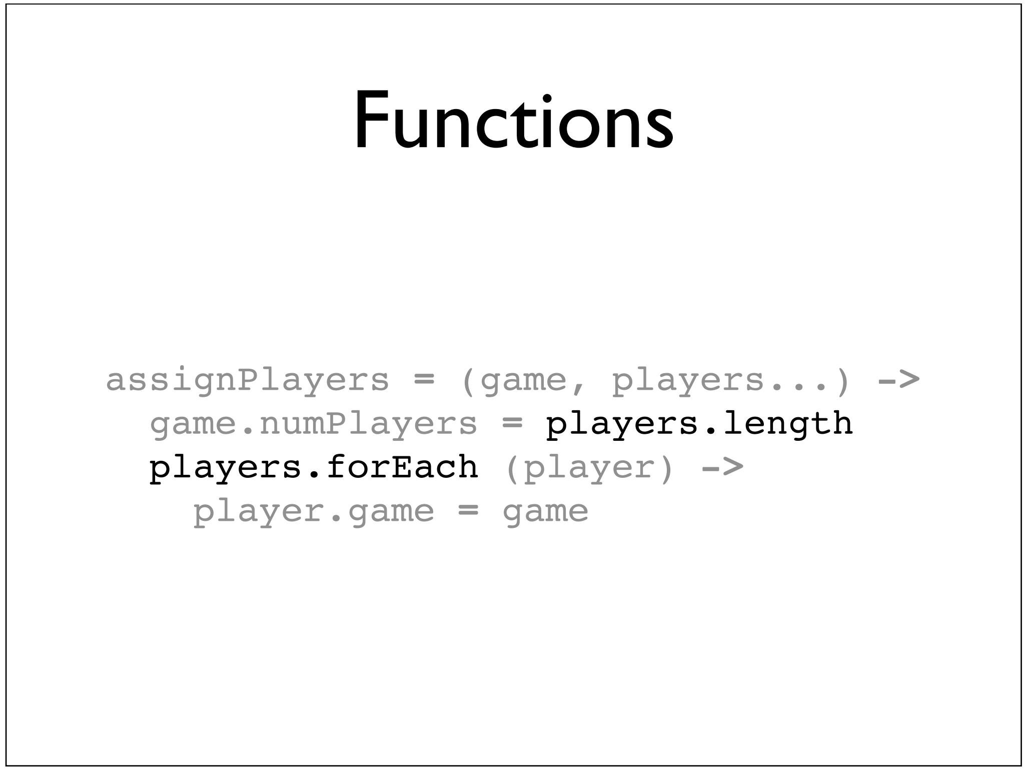
variable arguments
in js clunky
arguments is no array
CS: syntax sugar



Functions

```
assignPlayers = (game, players...) ->
  game.numPlayers = players.length
  players.forEach (player) ->
    player.game = game
```

pass them with “...”



Functions

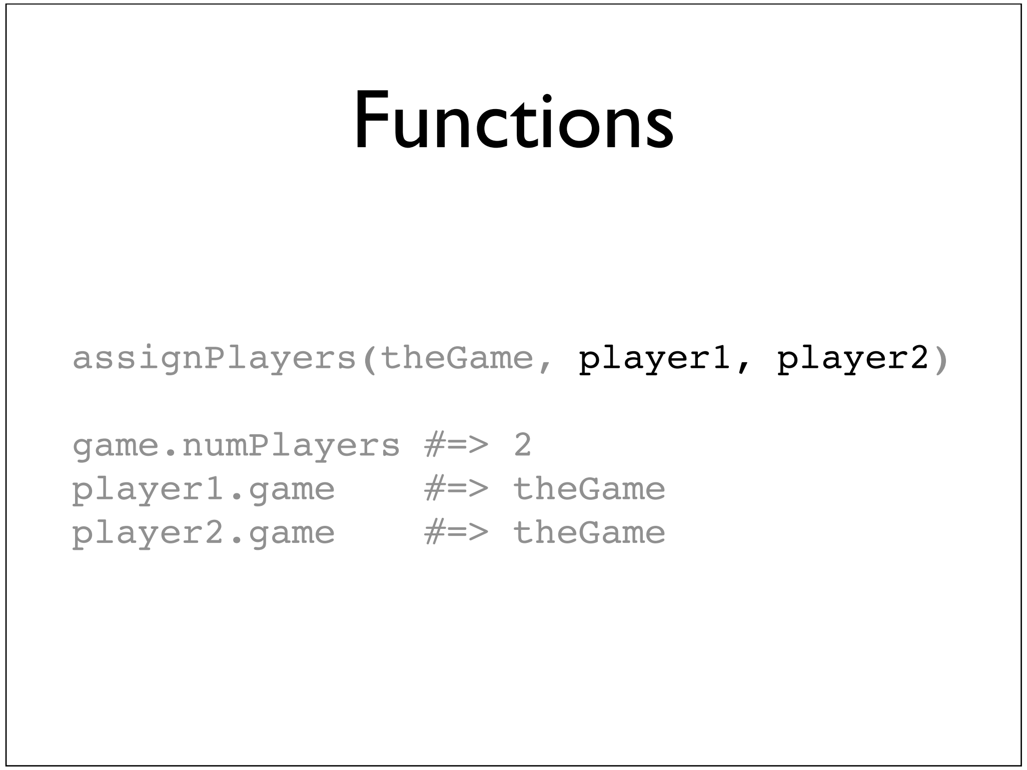
```
assignPlayers = (game, players...) ->
  game.numPlayers = players.length
  players.forEach (player) ->
    player.game = game
```

we get an array in the func

Functions

```
assignPlayers(theGame, player1, player2)

game.numPlayers #=> 2
player1.game    #=> theGame
player2.game    #=> theGame
```



Functions

```
assignPlayers(theGame, player1, player2)

game.numPlayers #=> 2
player1.game    #=> theGame
player2.game    #=> theGame
```

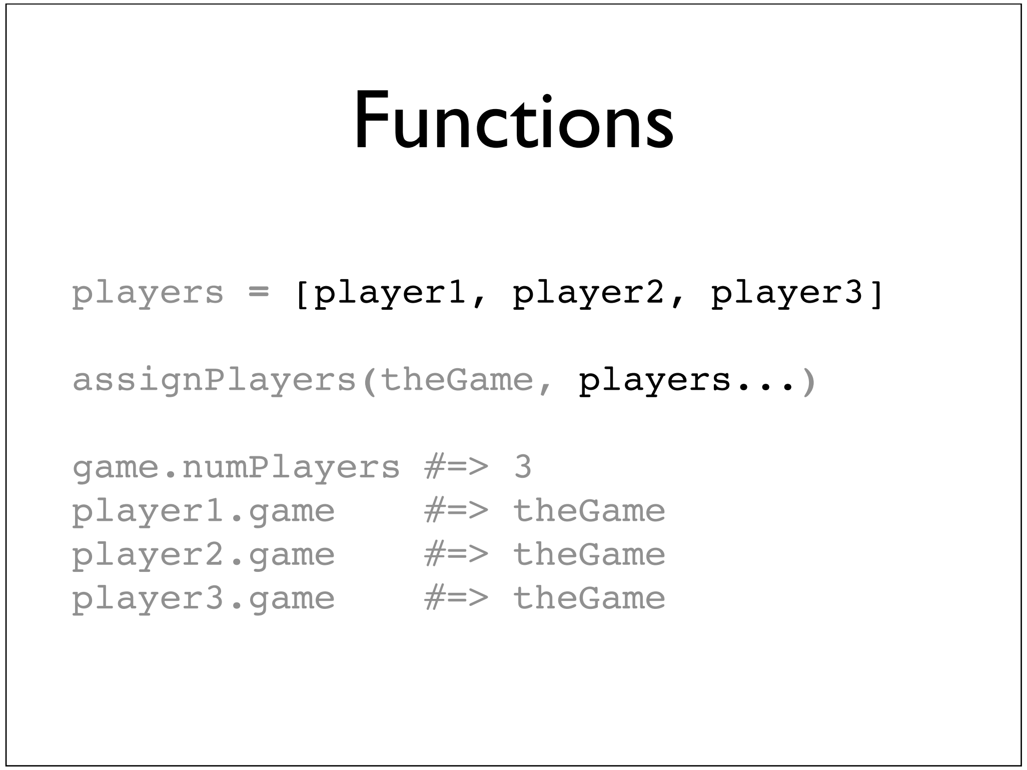
pass arguments, not an array

Functions

```
players = [player1, player2, player3]

assignPlayers(theGame, players...)

game.numPlayers #=> 3
player1.game    #=> theGame
player2.game    #=> theGame
player3.game    #=> theGame
```



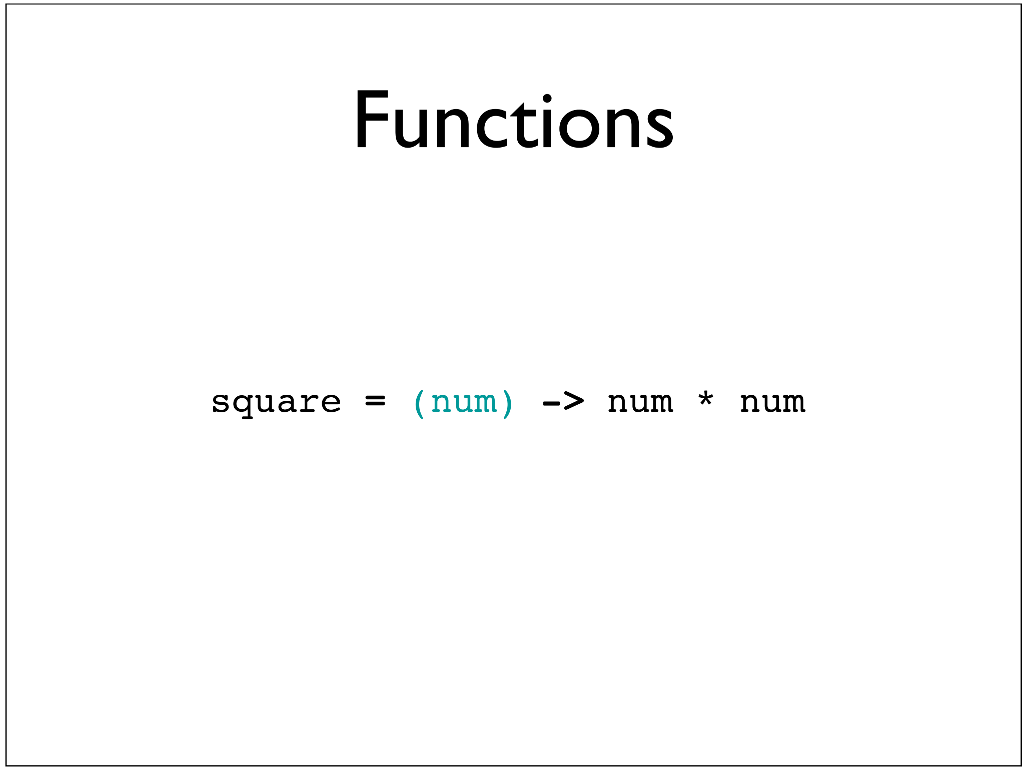
Functions

```
players = [player1, player2, player3]

assignPlayers(theGame, players...)

game.numPlayers #=> 3
player1.game    #=> theGame
player2.game    #=> theGame
player3.game    #=> theGame
```

you can also pass an explicit array
same syntax
internally Function.proto.apply



Functions

```
square = (num) -> num * num
```

comfty from ruby
lack of parens

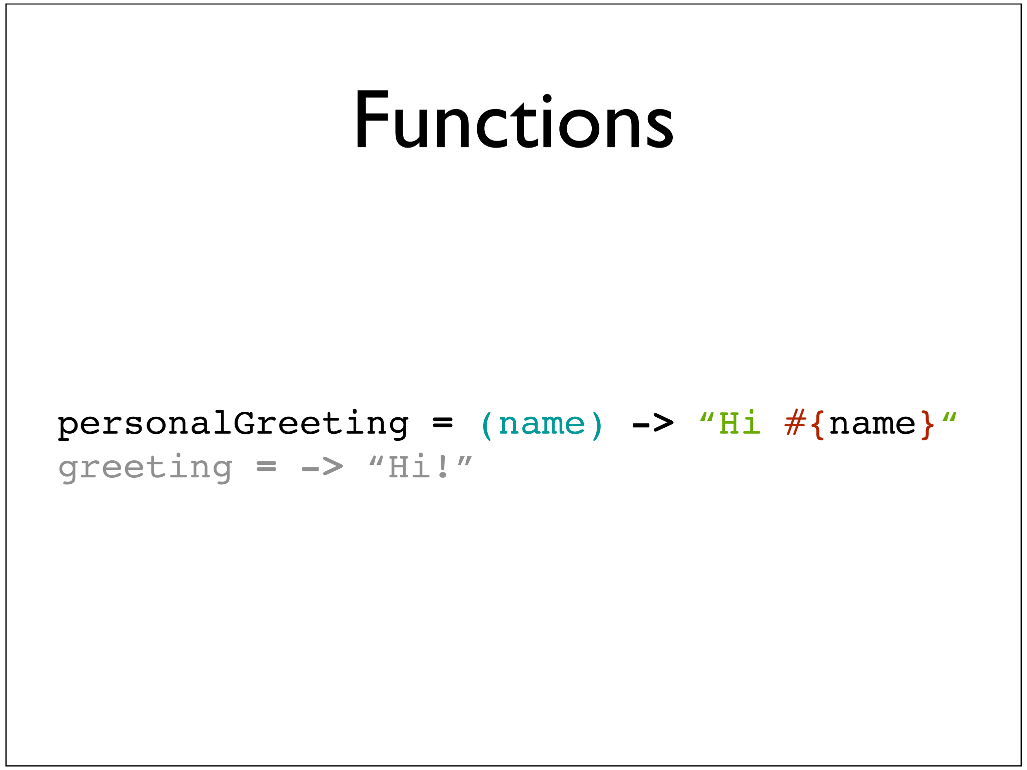
Functions

```
square = (num) -> num * num

square 2 #=> 4
```

Functions

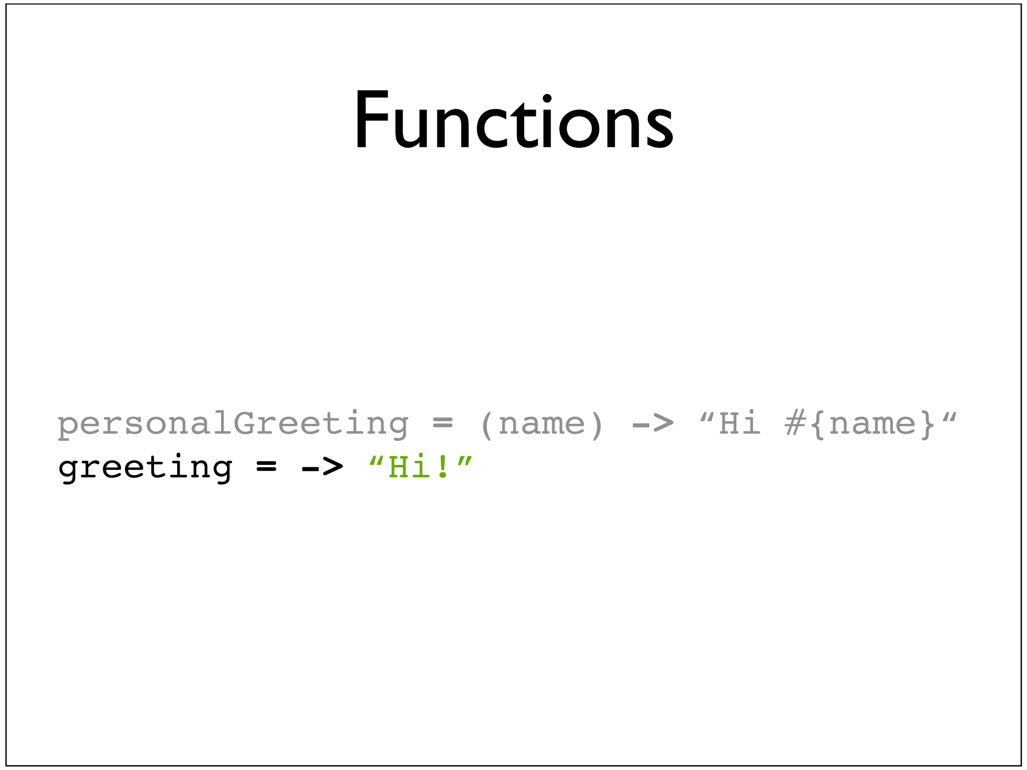
```
personalGreeting = (name) -> “Hi #{name}“
greeting = -> “Hi!”
```



Functions

```
personalGreeting = (name) -> “Hi #{name}“
greeting = -> “Hi!”
```

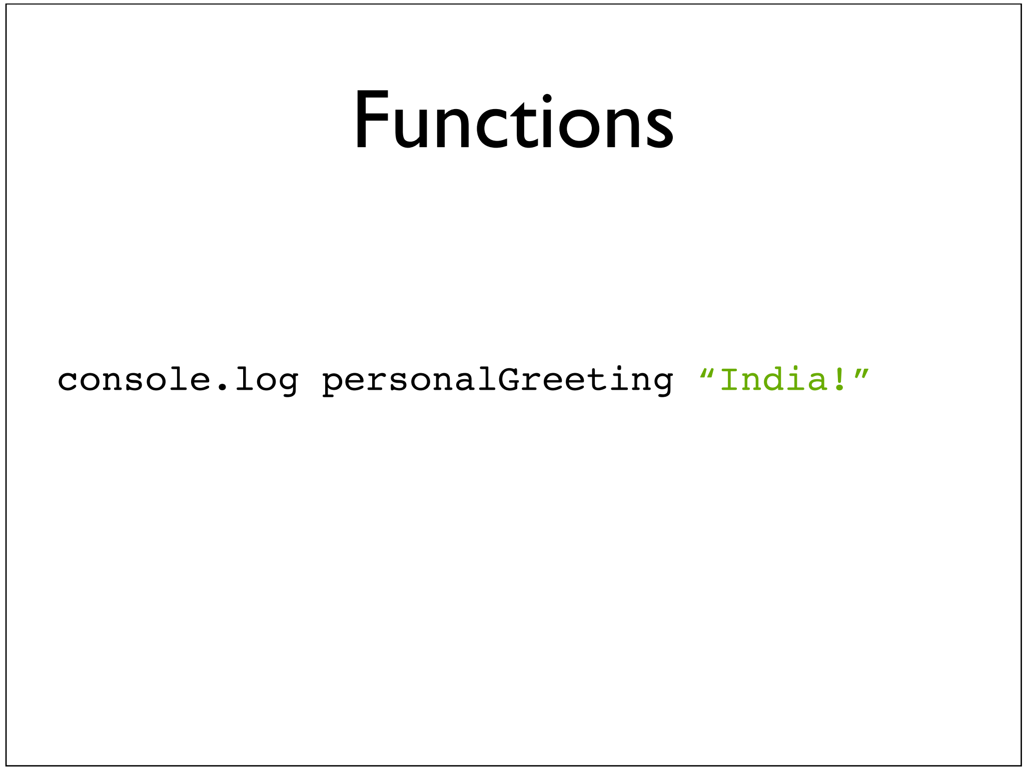
comfty from ruby
lack of parens



Functions

```
personalGreeting = (name) -> "Hi #{name}"
greeting = -> "Hi!"
```

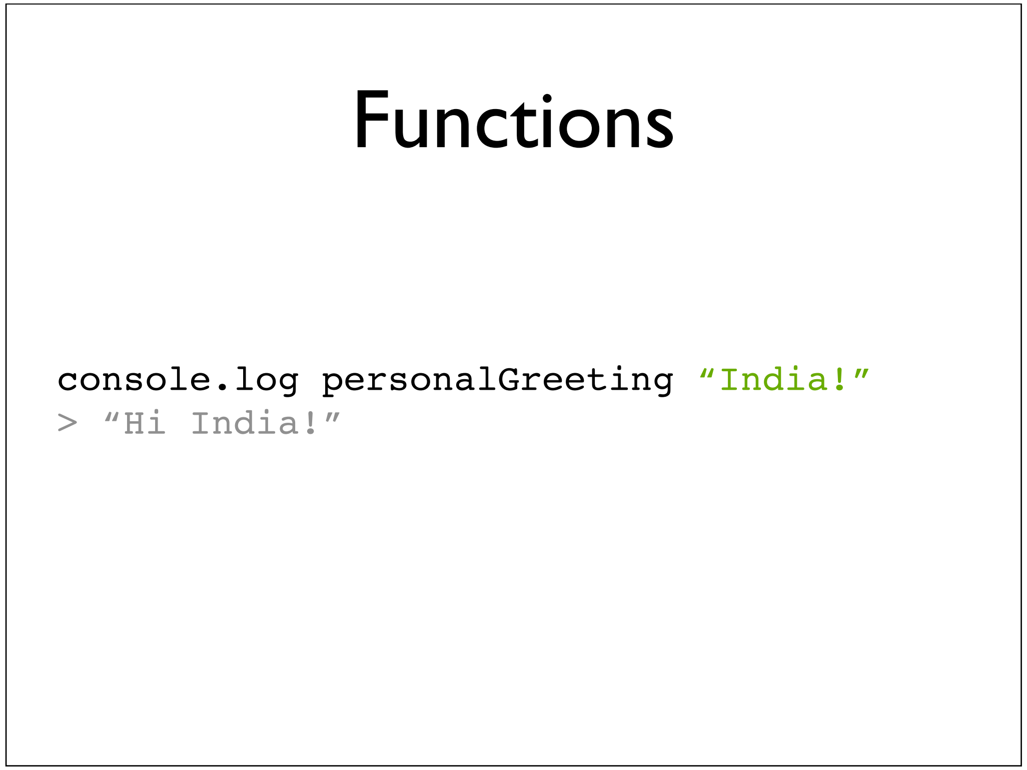
comfty from ruby
lack of parens



Functions

```
console.log personalGreeting “India!”
```

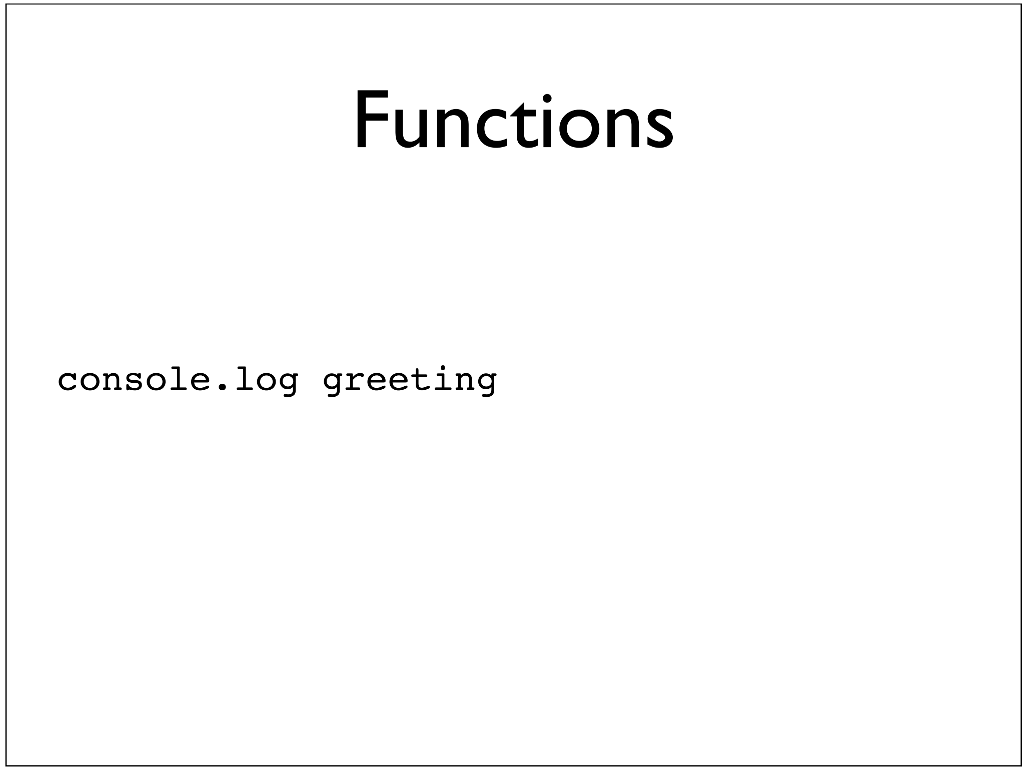
comfty from ruby
lack of parens



Functions

```
console.log personalGreeting “India!”
> “Hi India!”
```

comfty from ruby
lack of parens



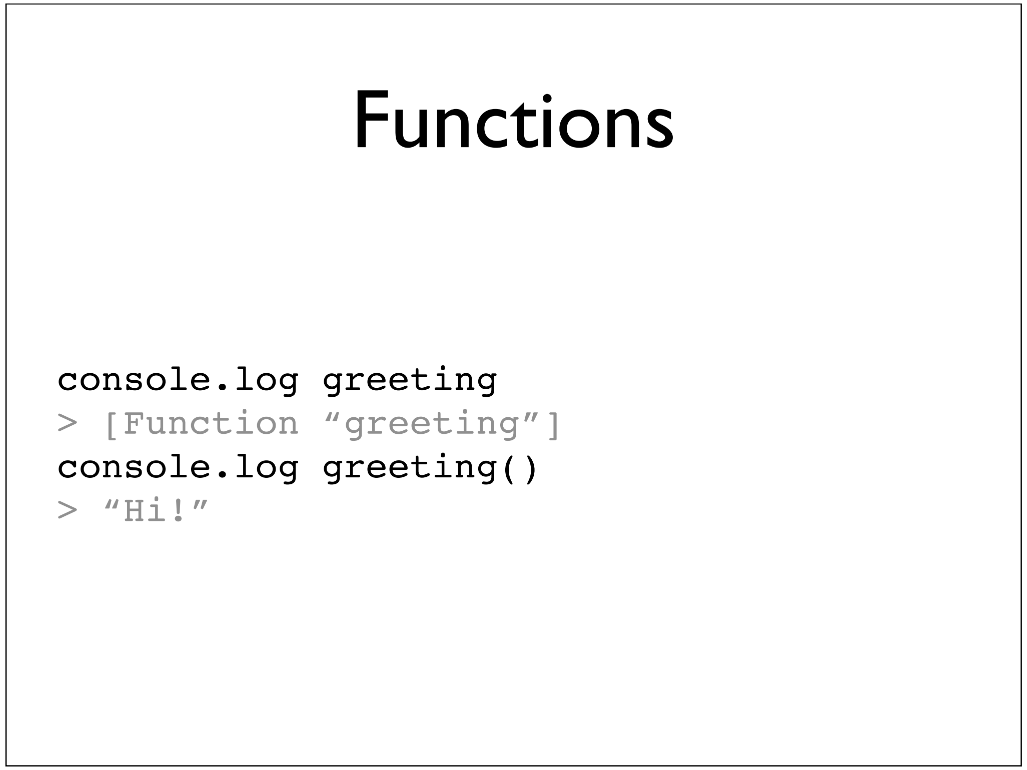
Functions

```
console.log greeting
```

what do you expect?

Functions

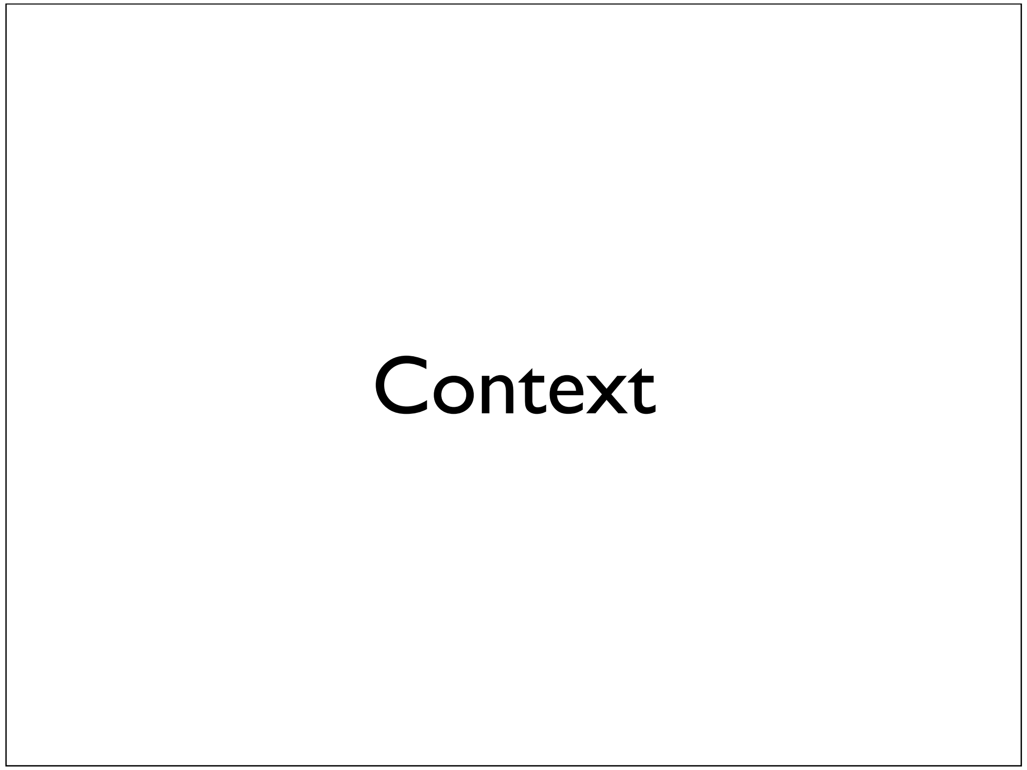
```
console.log greeting
> [Function “greeting”]
```



Functions

```
console.log greeting
> [Function “greeting”]
console.log greeting()
> “Hi!”
```

what do you expect?



Context

javascript has keyword “this”
it refers to current “evaluation context”
akin to current scope

Context

```
User = (id, name) ->
  this.id = id
  this.name = name
  this.element = $(“#user-#{this.id}”)
```

Context

```
User = (id, name) ->
  this.id = id
  this.name = name
  this.element = $(“#user-#{this.id}”)
```

Context

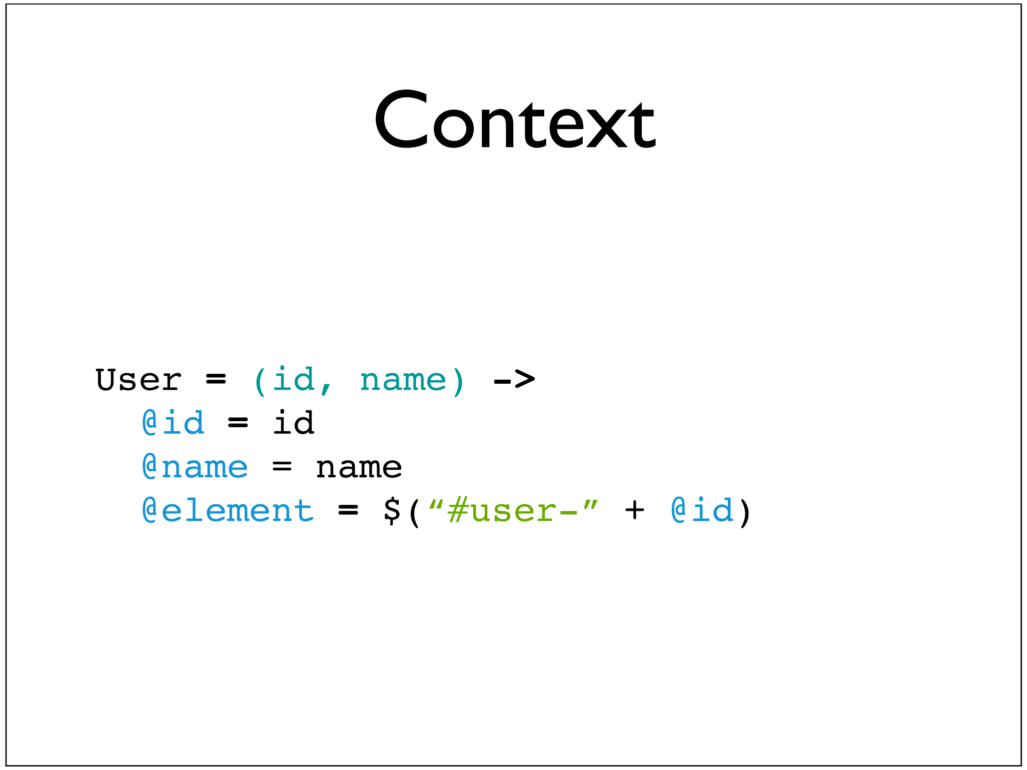
```
User = (id, name) ->
  this.id = id
  this.name = name
  this.element = $(“#user-#{this.id}”)
```

Context

```
User = (id, name) ->
  this.id = id
  this.name = name
  this.element = $(“#user-#{this.id}”)
```

Context

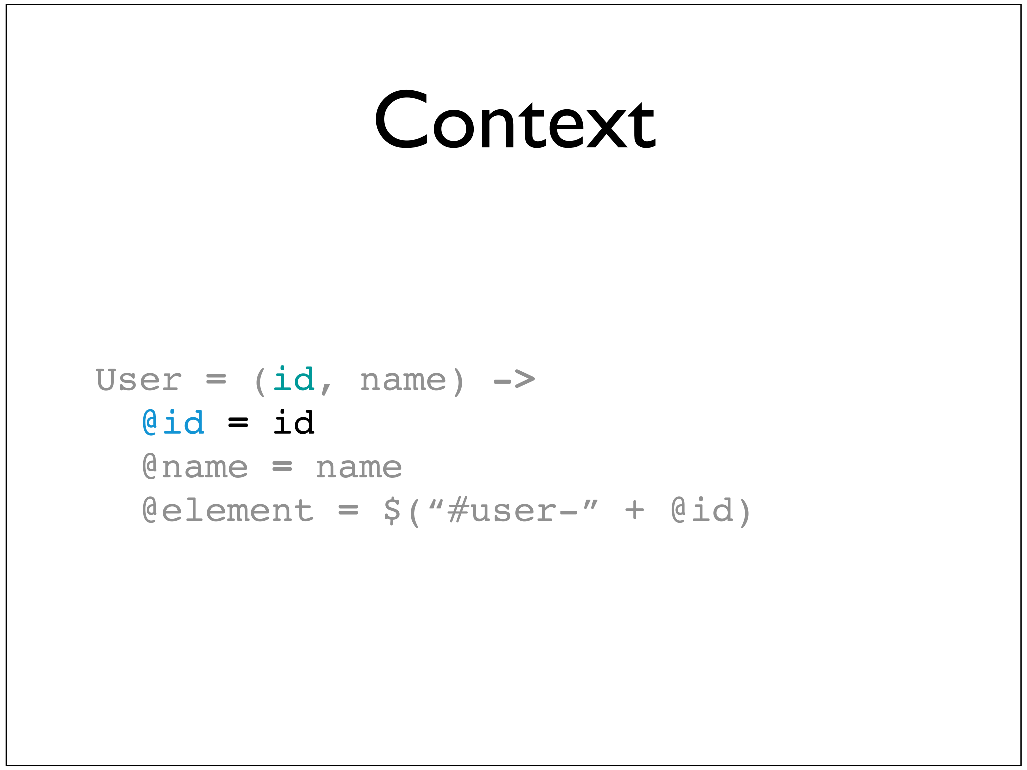
```
User = (id, name) ->
  this.id = id
  this.name = name
  this.element = $(“#user-#{this.id}”)
```



Context

```
User = (id, name) ->
  @id = id
  @name = name
  @element = $("#user-" + @id)
```

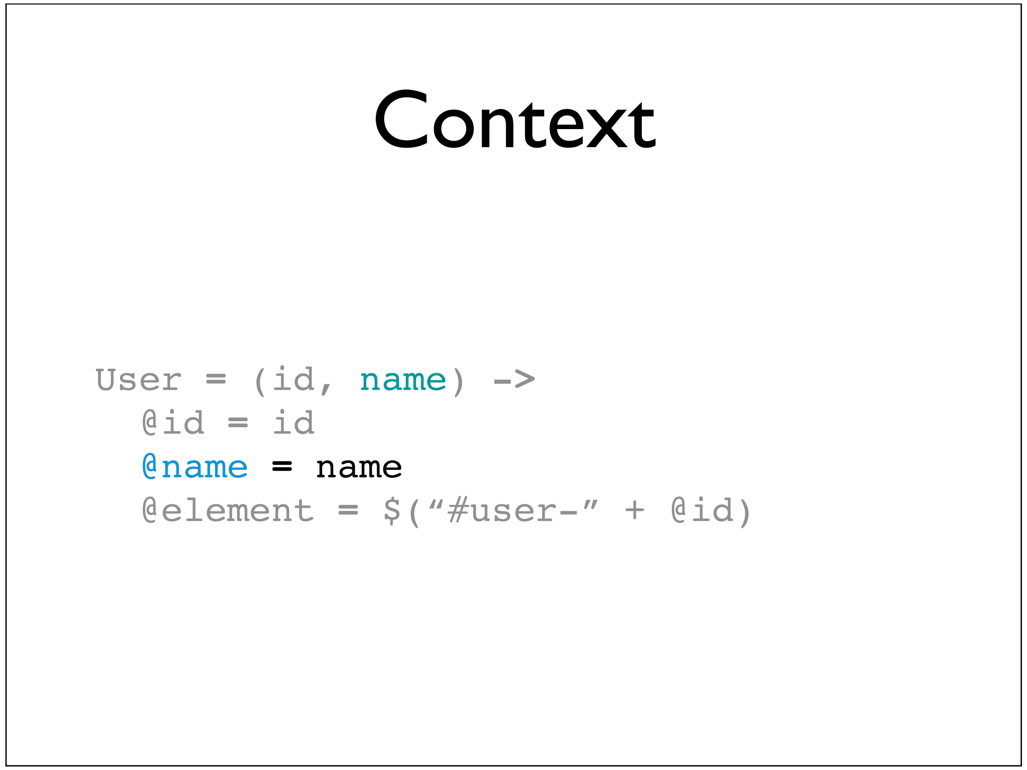
there's a bit of repetition



Context

```
User = (id, name) ->
  @id = id
  @name = name
  @element = $(“#user-” + @id)
```

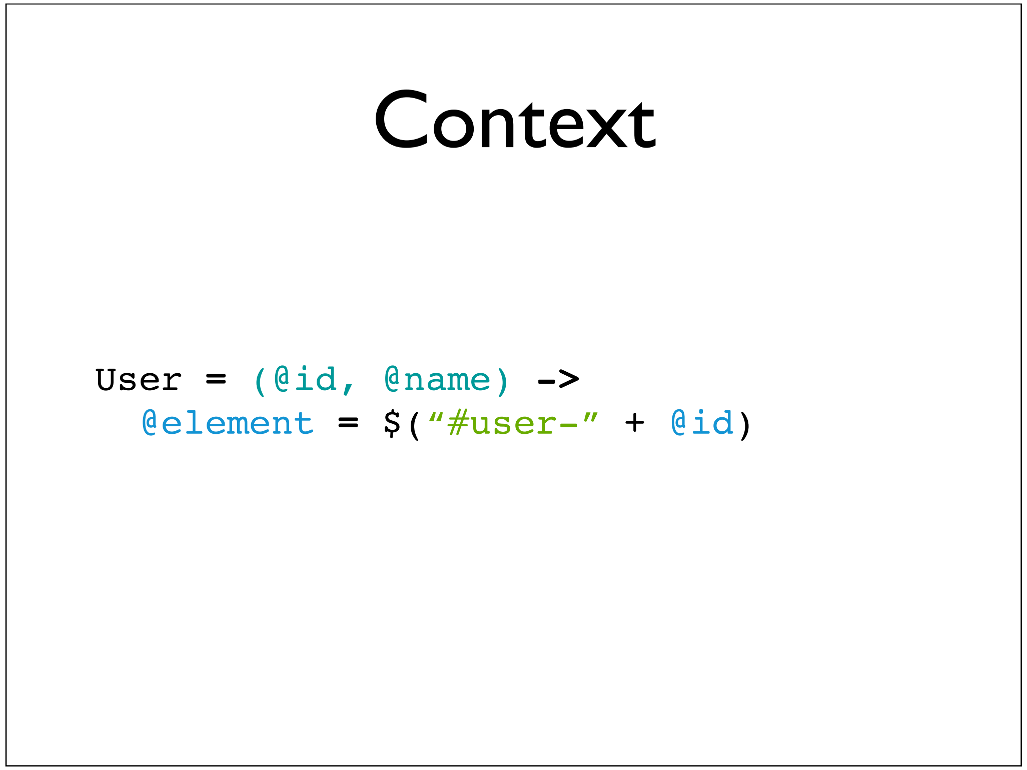
define arguments…



Context

```
User = (id, name) ->
  @id = id
  @name = name
  @element = $(“#user-” + @id)
```

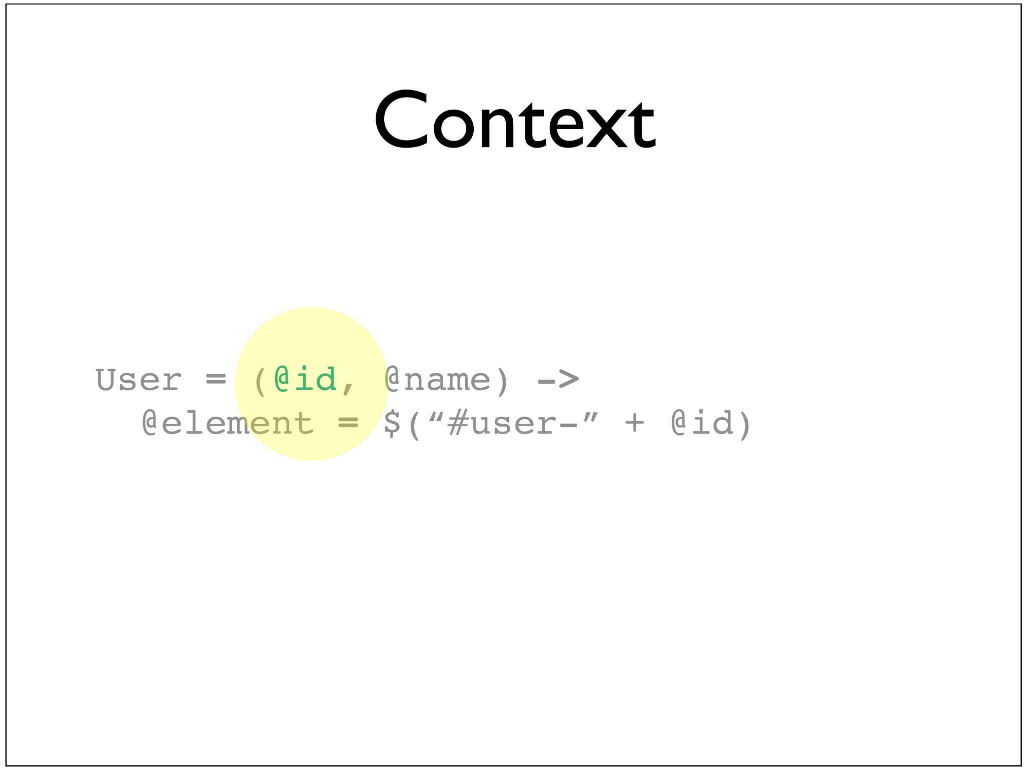
… just to assign them to this.whatever



Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)
```

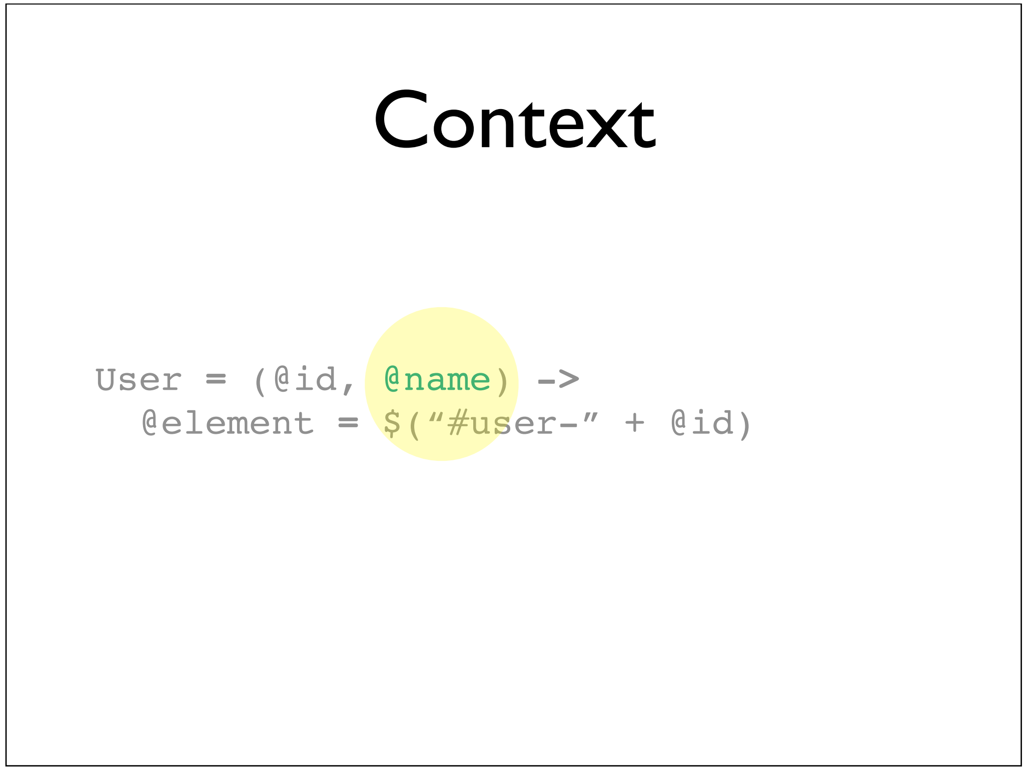
shorthand syntax



Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)
```

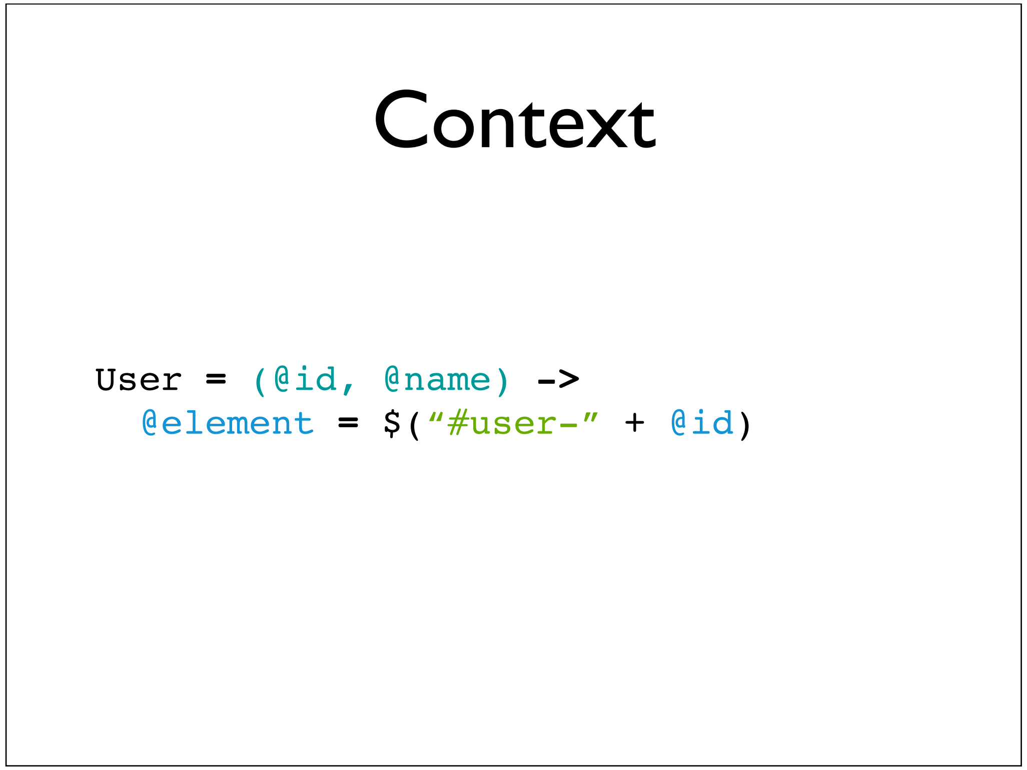
assigns the first argument to this.id



Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)
```

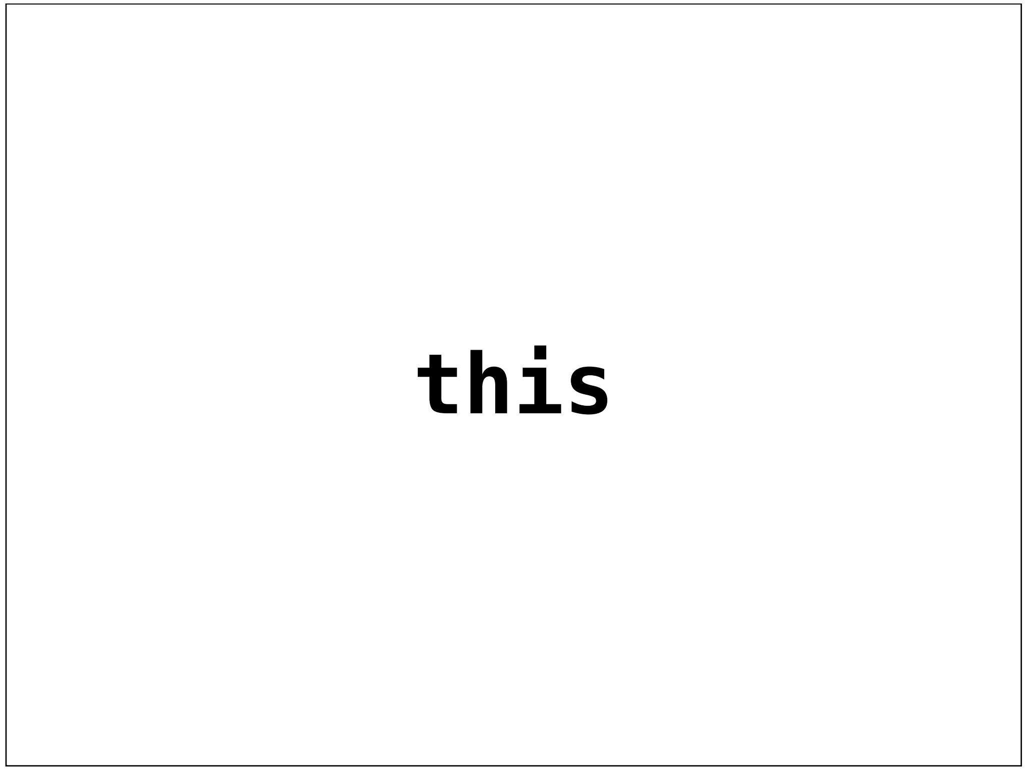
assigns the second argument to this.name



Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)
```

not the most useful
but it's used a lot
so worth mentioning



`this`

…we need to talk a bit about this

this

```
var Cat = {
  cuteness: 10,
  lovable: function() {
    return this.cuteness >= 5
  }
}

Cat.lovable() #=> true
```

this

```
var Cat = {
  cuteness: 10,
  lovable: function() {
    return this.cuteness >= 5
  }
}

Cat.lovable() #=> true
```

this

```
var Cat = {
  cuteness: 10,
  lovable: function() {
    return this.cuteness >= 5
  }
}

Cat.lovable() #=> true
```

this

```
var Cat = {
  cuteness: 10,
  lovable: function() {
    return this.cuteness >= 5
  }
}

Cat.lovable() #=> true
```

this

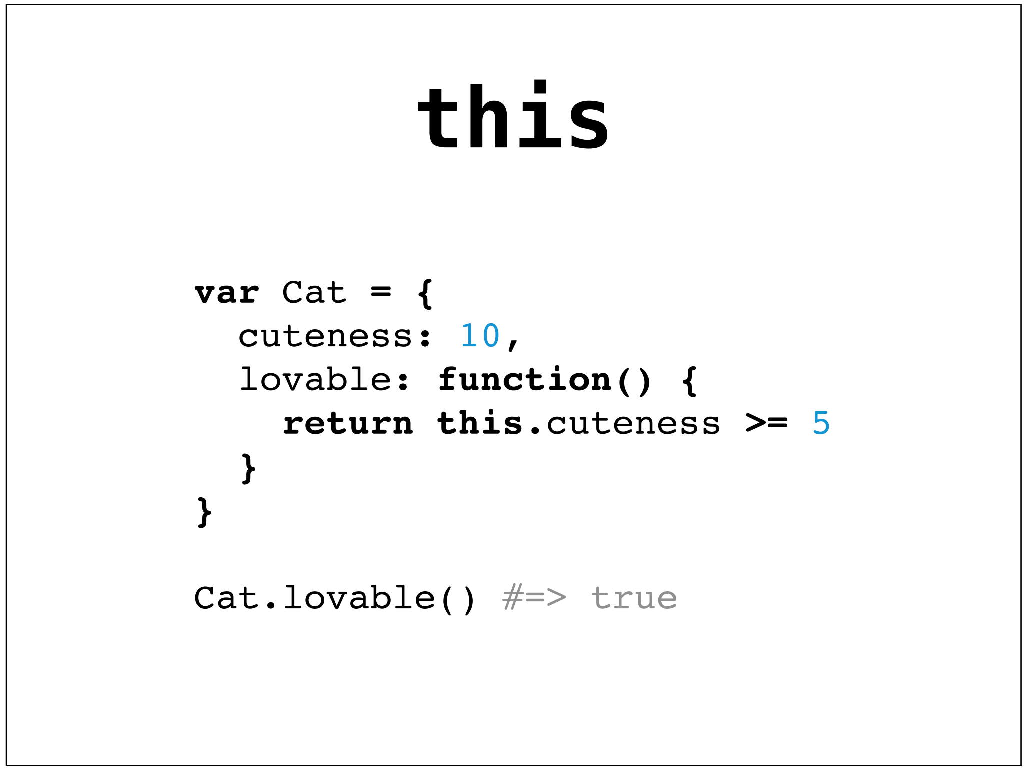
```
var Cat = {
  cuteness: 10,
  lovable: function() {
    return this.cuteness >= 5
  }
}

Cat.lovable() #=> true
```

this

```
var Cat = {
  cuteness: 10,
  lovable: function() {
    return this.cuteness >= 5
  }
}

Cat.lovable() #=> true
```



this

```
var Cat = {
  cuteness: 10,
  lovable: function() {
    return this.cuteness >= 5
  }
}

Cat.lovable() #=> true
```

not all cats are created equal
some aren’t as cute

not all cats are created equal
some aren’t as cute

not all cats are created equal
some aren’t as cute

this

```
function Cat(cuteness) {
  this.cuteness >= cuteness;
}
Cat.prototype.lovable = function() {
  return this.cuteness >= 5;
}
```

this

```
function Cat(cuteness) {
  this.cuteness >= cuteness;
}
Cat.prototype.lovable = function() {
  return this.cuteness >= 5;
}
```

this

```
function Cat(cuteness) {
  this.cuteness >= cuteness;
}
Cat.prototype.lovable = function() {
  return this.cuteness >= 5;
}
```

this

```
var cuteKitty = new Cat(10)
var meanCat = new Cat(2)
```

this

```
var cuteKitty = new Cat(10)
var meanCat = new Cat(2)
```

this

```
var cuteKitty = new Cat(10)
var meanCat = new Cat(2)
```

this

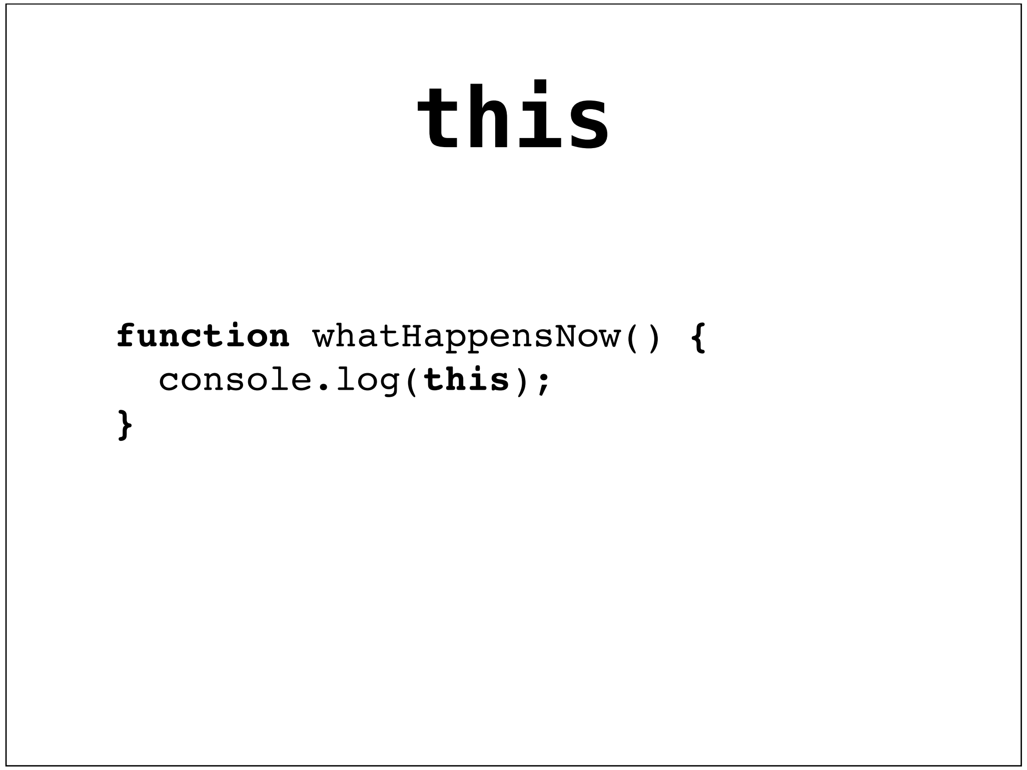
```
var cuteKitty = new Cat(10)
var meanCat = new Cat(2)

cuteKitty.lovable() #=> true
```

this

```
var cuteKitty = new Cat(10)
var meanCat = new Cat(2)

cuteKitty.lovable() #=> true
meanCat.lovable() #=> false
```



this

```
function whatHappensNow() {
  console.log(this);
}
```

can anyone tell?

this

```
function whatHappensNow() {
  console.log(this);
}

whatHappensNow()
> [window]
```

this

```
$(“.user a.delete”).click(function() {
  console.log(this);
  $.ajax(...)
});
```

this

```
$(“.user a.delete”).click(function() {
  console.log(this);
  $.ajax(...)
});
```

this

```
$(".user a.delete").click(function() {
  console.log(this);
  $.ajax(...)
});

<a href="#" class="delete">Delete</a>
```

Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)
```

let’s go back to our User

Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)

User.prototype.destroy = ->
  $.ajax(...)
```

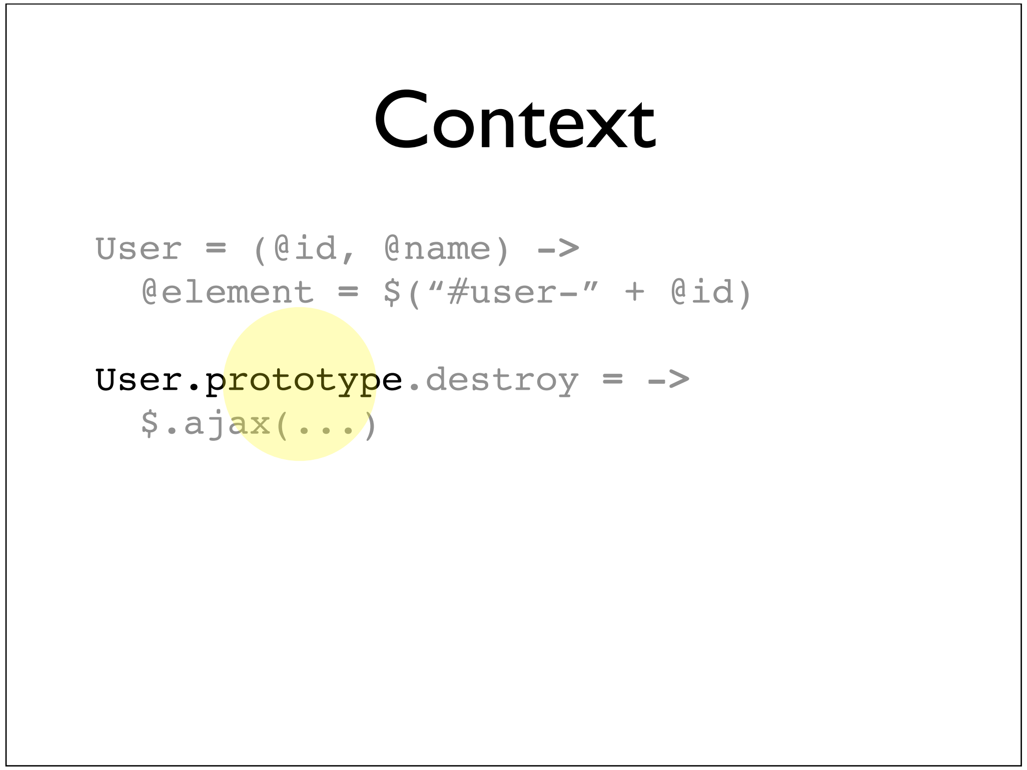
let’s say user has destroy fn

Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)

User.prototype.destroy = ->
  $.ajax(...)
```

triggers ajax request
we don’t care about internals

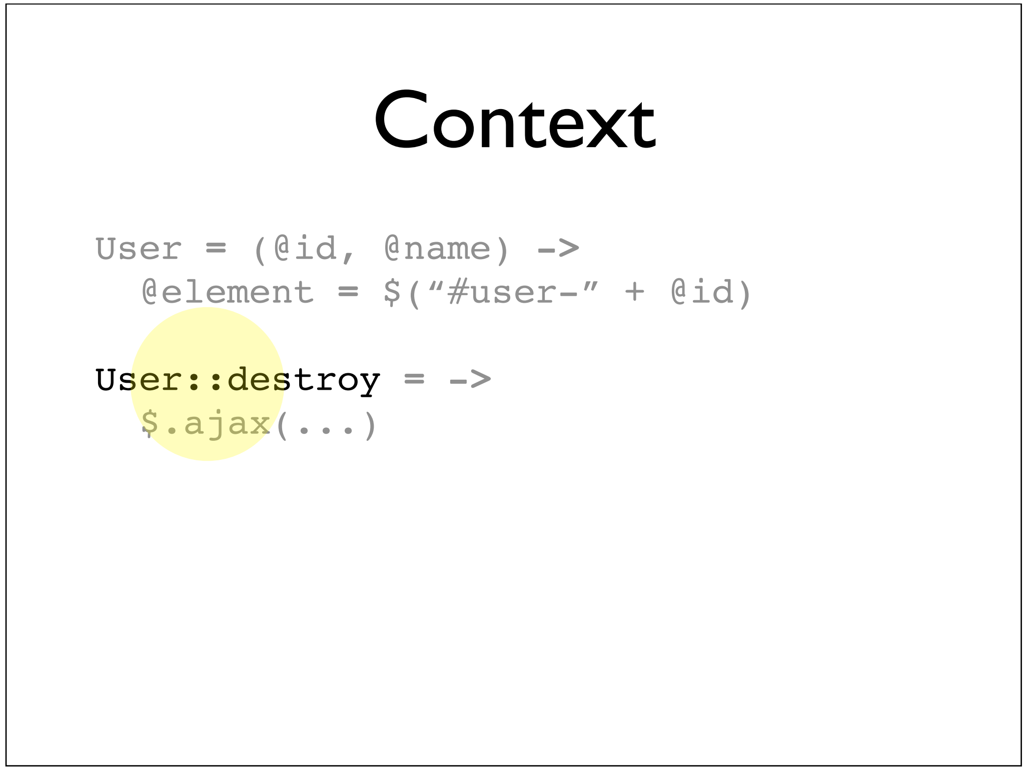


Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)

User.prototype.destroy = ->
  $.ajax(...)
```

User::destroy, User::save, User blah blah

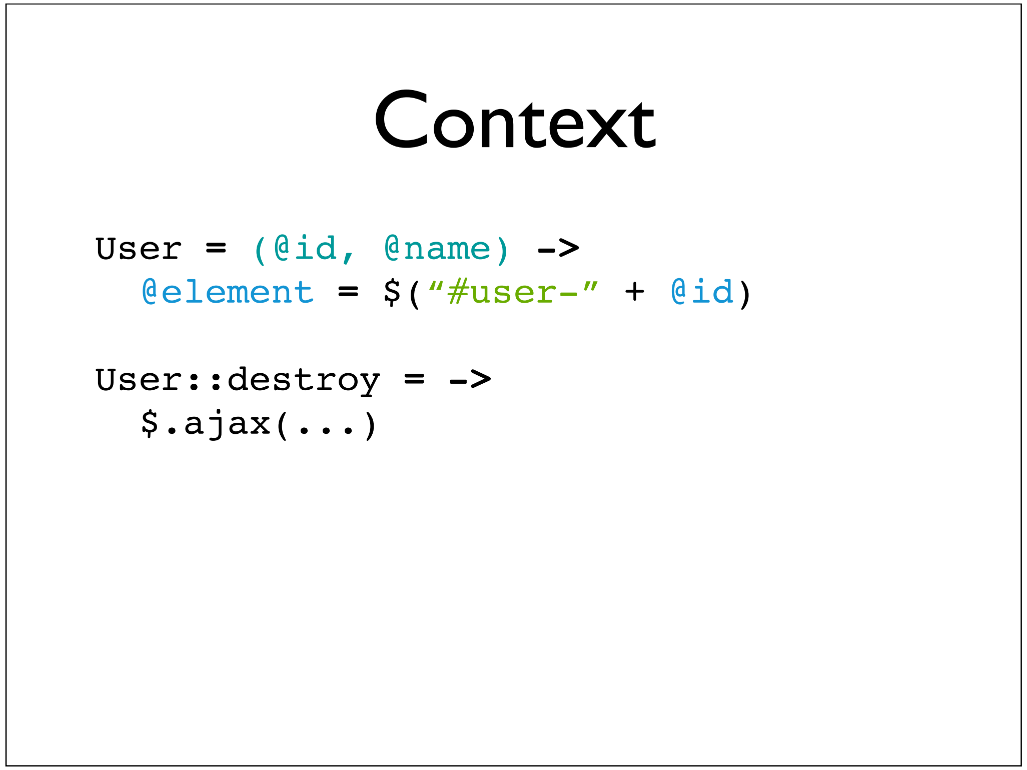


Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)

User::destroy = ->
  $.ajax(...)
```

shorthand syntax



Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)

User::destroy = ->
  $.ajax(...)
```

trigger by a browser event

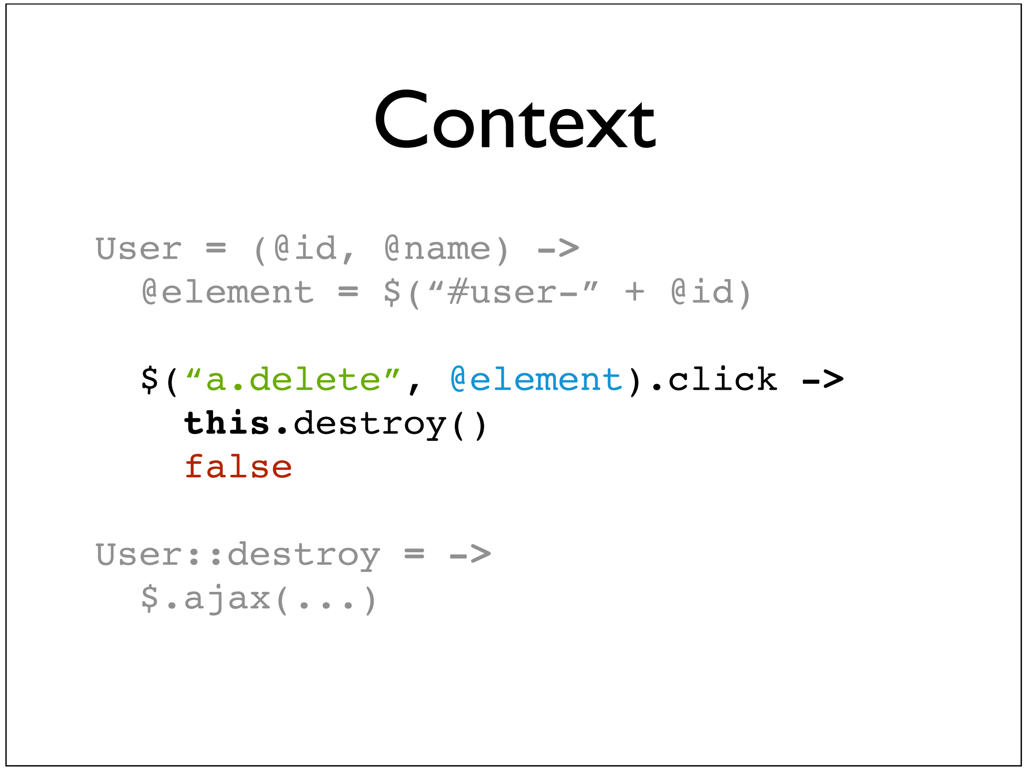
Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)

  $("a.delete", @element).click ->
    this.destroy()
    false

User::destroy = ->
  $.ajax(...)
```

naive implementation



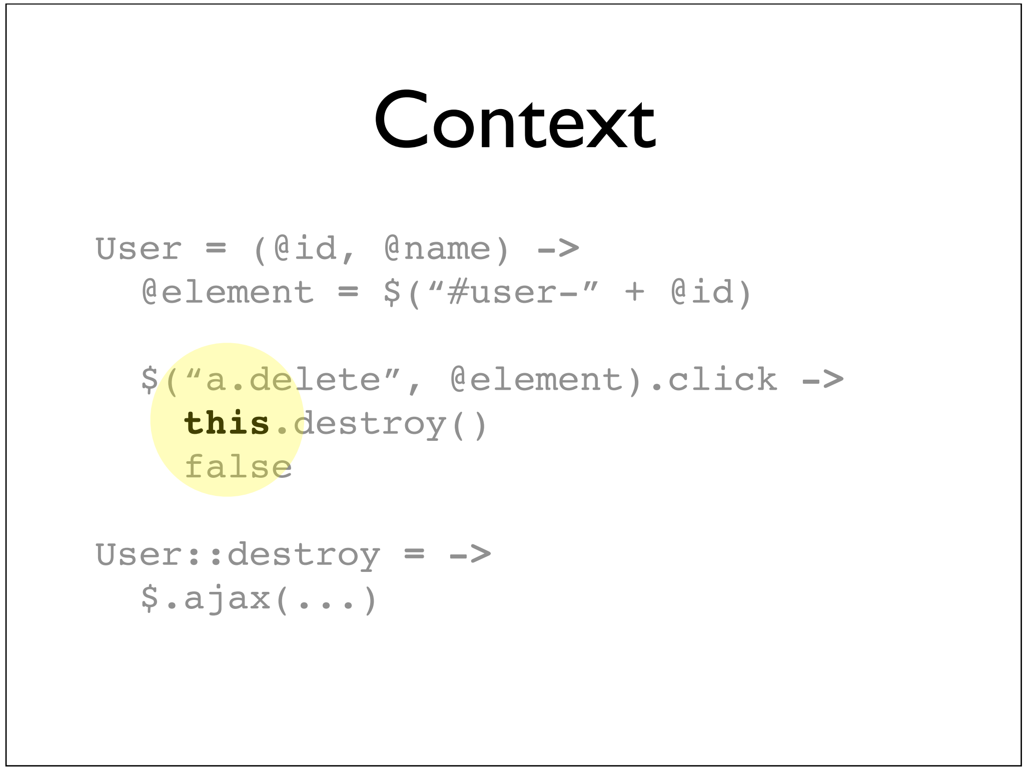
Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)

  $("a.delete", @element).click ->
    this.destroy()
    false

User::destroy = ->
  $.ajax(...)
```

naive implementation
doesn’t work



Context

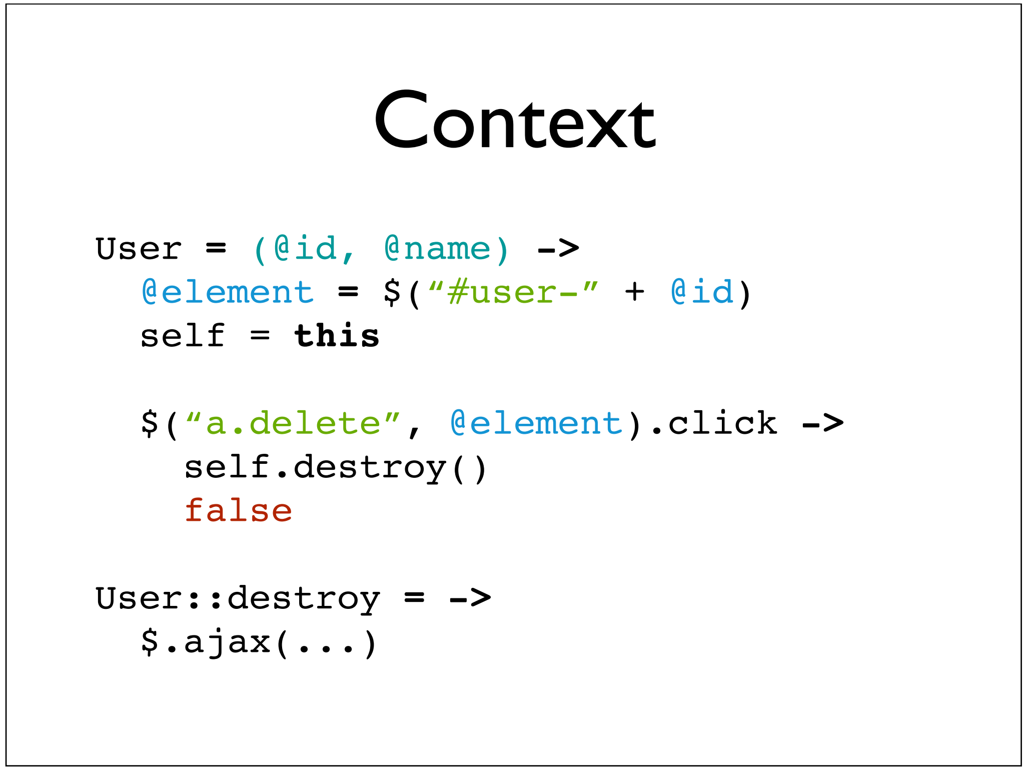
```
User = (@id, @name) ->
  @element = $("#user-" + @id)

  $("a.delete", @element).click ->
    this.destroy()
    false

User::destroy = ->
  $.ajax(...)
```

dom element

jquery object



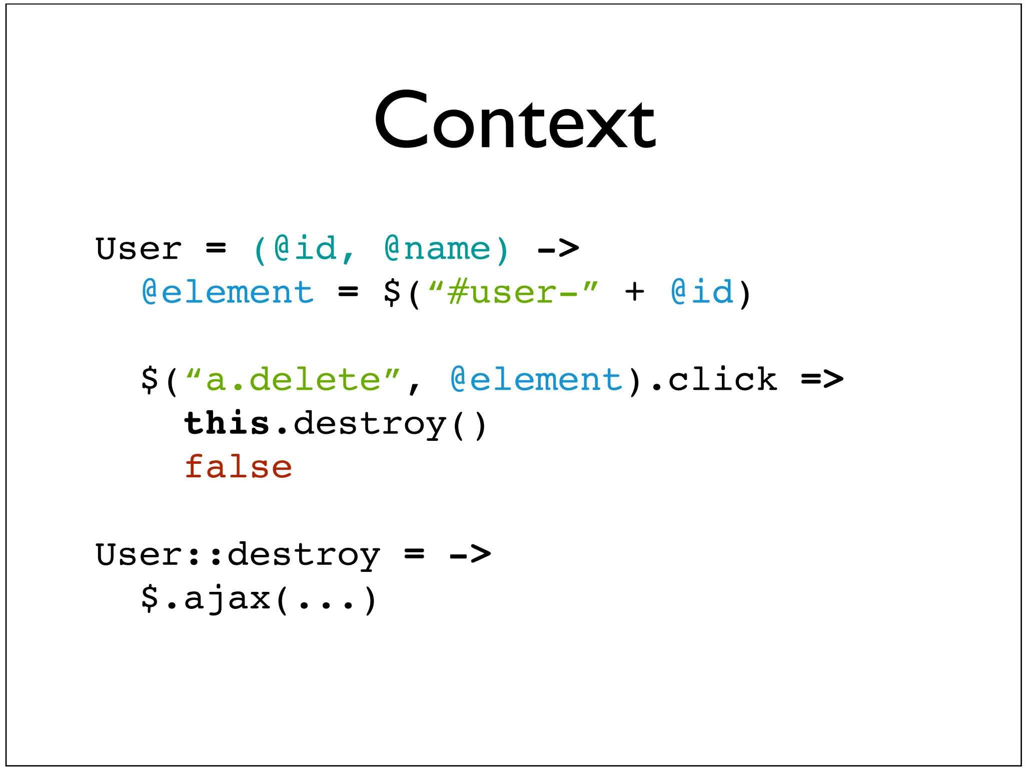
Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)
  self = this

  $("a.delete", @element).click ->
    self.destroy()
    false

User::destroy = ->
  $.ajax(...)
```

naive solution



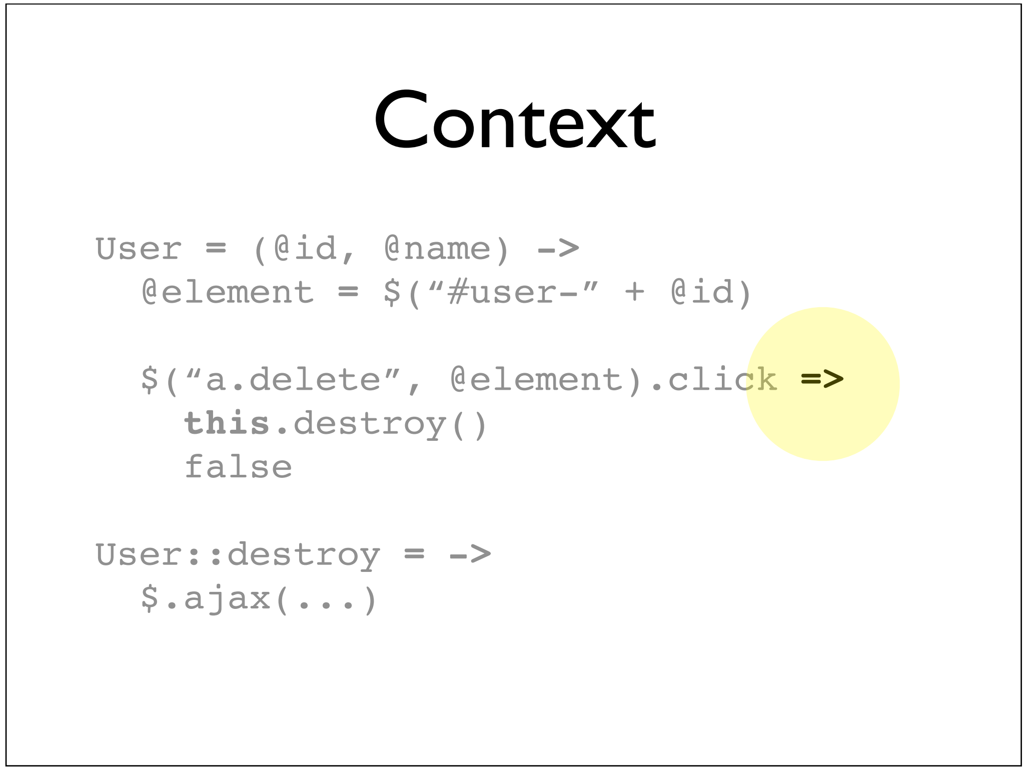
Context

```
User = (@id, @name) ->
  @element = $("#user-" + @id)

  $("a.delete", @element).click =>
    this.destroy()
    false

User::destroy = ->
  $.ajax(...)
```

better solution



Context

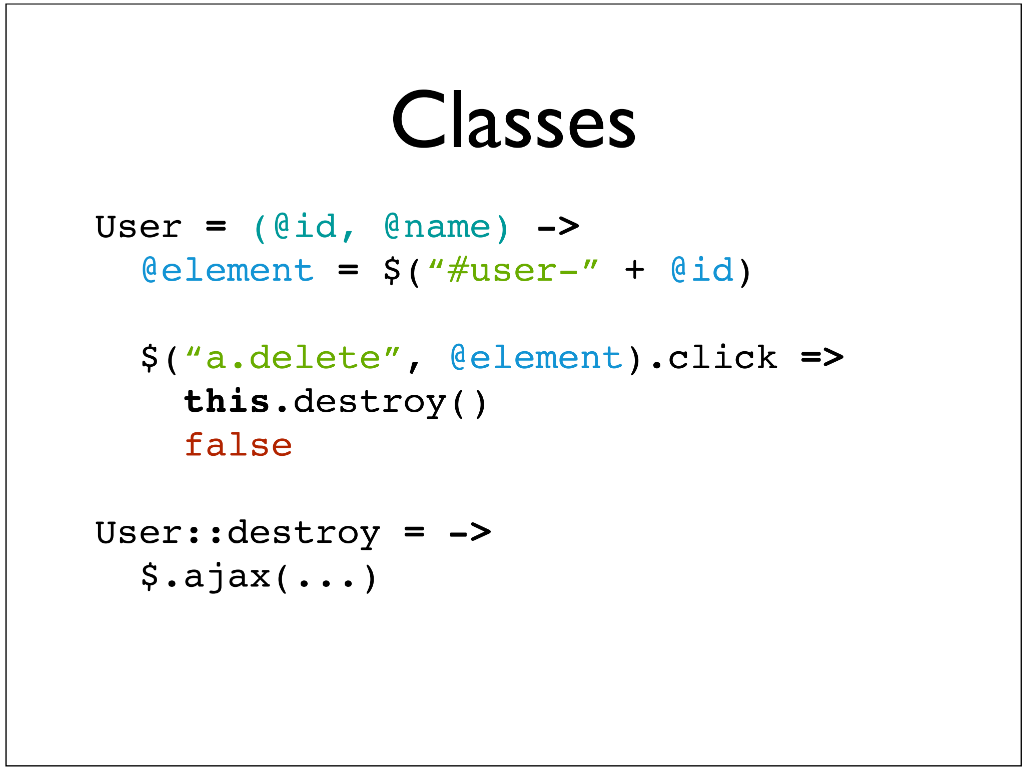
```
User = (@id, @name) ->
  @element = $("#user-" + @id)

  $("a.delete", @element).click =>
    this.destroy()
    false

User::destroy = ->
  $.ajax(...)
```

better solution
forces this inside function

Classes



Classes

```
User = (@id, @name) ->
  @element = $("#user-" + @id)

  $("a.delete", @element).click =>
    this.destroy()
    false

User::destroy = ->
  $.ajax(...)
```

pretty, but not idiomatic
this looks a lot like a class

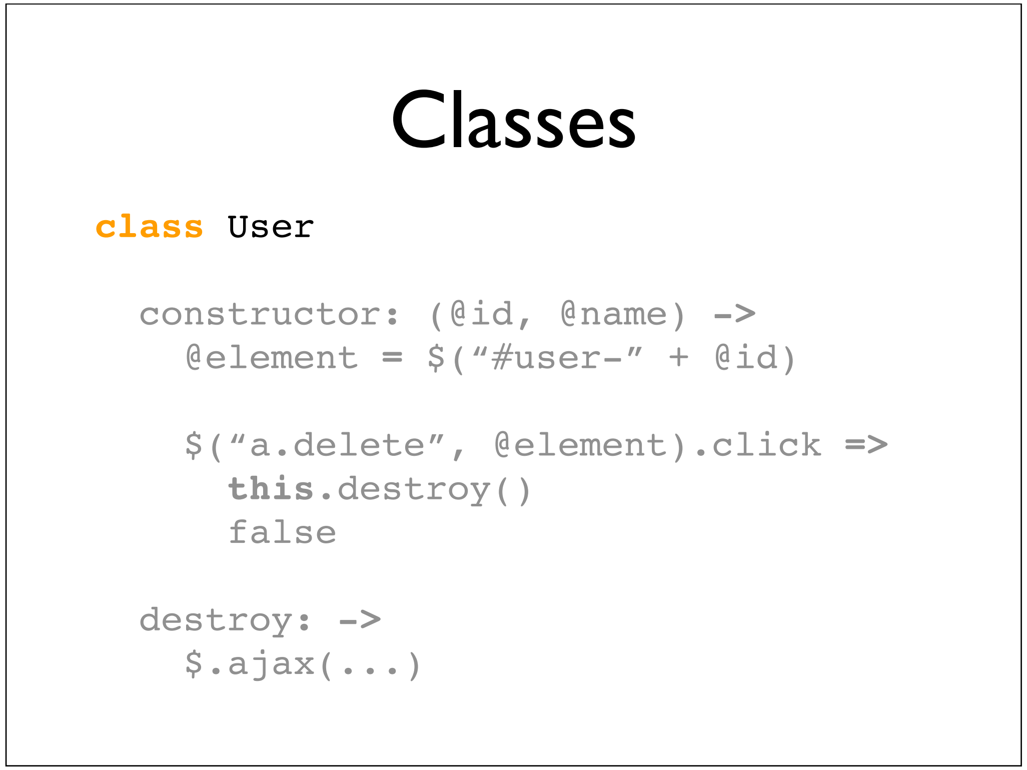
Classes

```
class User

  constructor: (@id, @name) ->
    @element = $("#user-" + @id)

    $("a.delete", @element).click =>
      this.destroy()
      false

  destroy: ->
    $.ajax(...)
```



Classes

```
class User

  constructor: (@id, @name) ->
    @element = $(“#user-” + @id)

    $(“a.delete”, @element).click =>
      this.destroy()
      false

  destroy: ->
    $.ajax(...)
```

class keyword

Classes

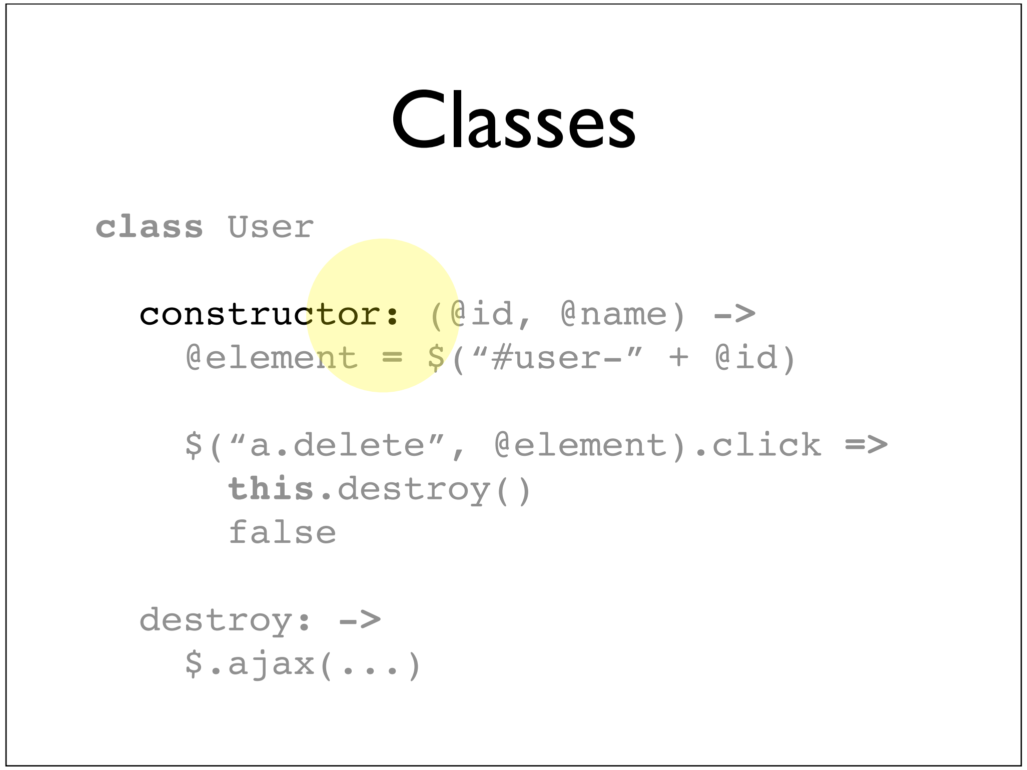
```
class User

  constructor: (@id, @name) ->
    @element = $("#user-" + @id)

    $("a.delete", @element).click =>
      this.destroy()
      false

  destroy: ->
    $.ajax(...)
```

constructor function



Classes

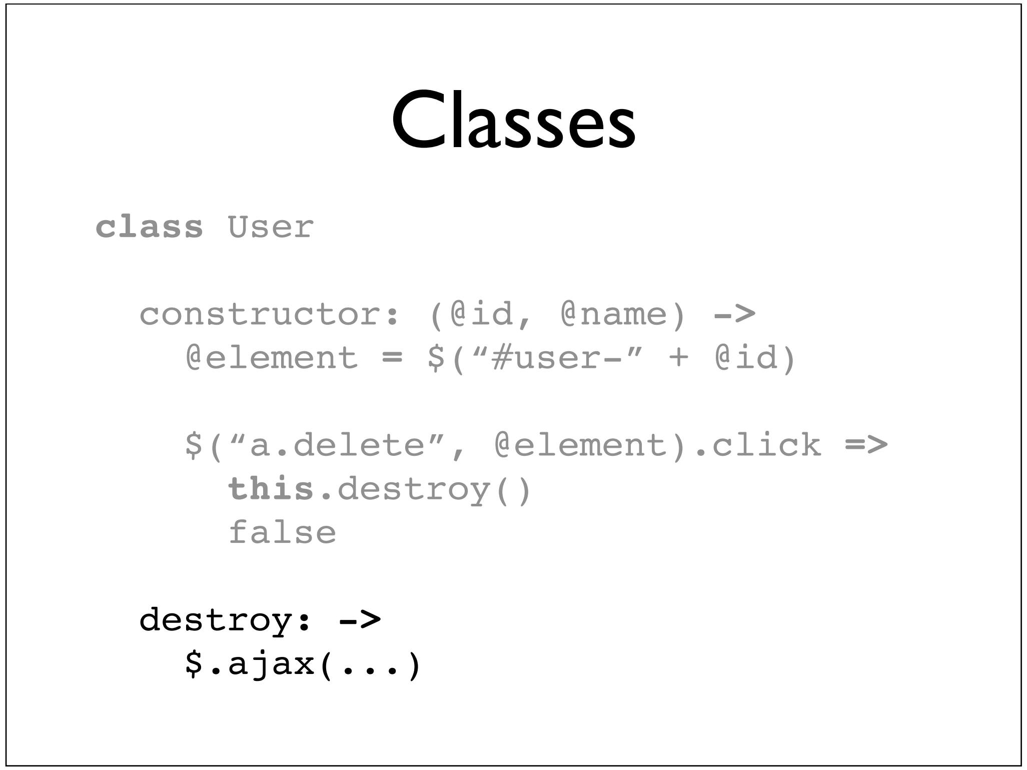
```
class User

  constructor: (@id, @name) ->
    @element = $("#user-" + @id)

    $("a.delete", @element).click =>
      this.destroy()
      false

  destroy: ->
    $.ajax(...)
```

note we use ":" for functions in a class



Classes

```
class User

  constructor: (@id, @name) ->
    @element = $("#user-" + @id)

    $("a.delete", @element).click =>
      this.destroy()
      false

  destroy: ->
    $.ajax(...)
```

other functions are just nested
no explicit prototypes
again, note :

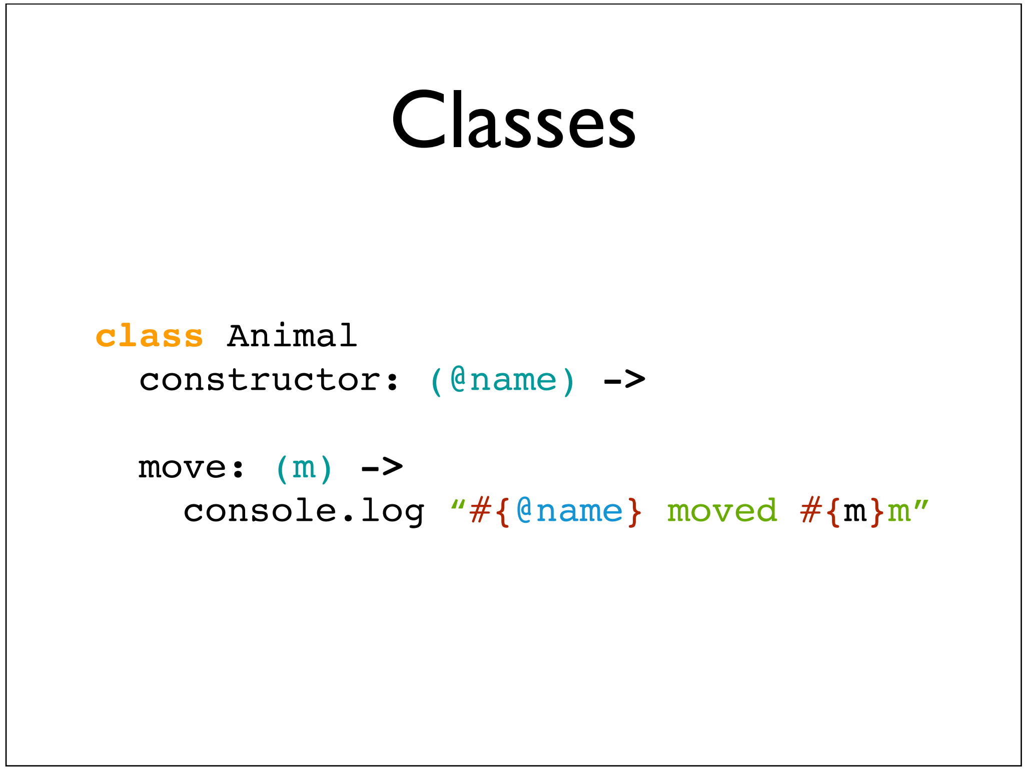
Classes

```
class User

  constructor: (@id, @name) ->
    @element = $("#user-" + @id)

    $("a.delete", @element).click =>
      this.destroy()
      false

  destroy: ->
    $.ajax(...)
```

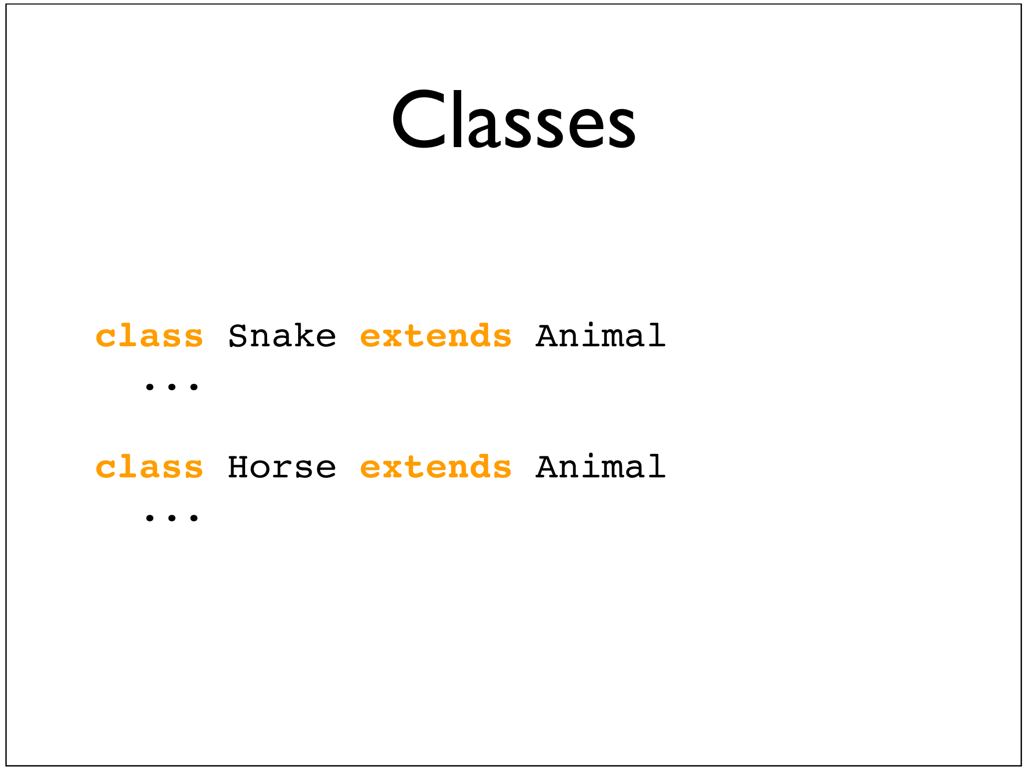


Classes

```
class Animal
  constructor: (@name) ->

  move: (m) ->
    console.log “#{@name} moved #{m}m”
```

another example
animal that moves



Classes

```
class Snake extends Animal
  ...

class Horse extends Animal
  ...
```

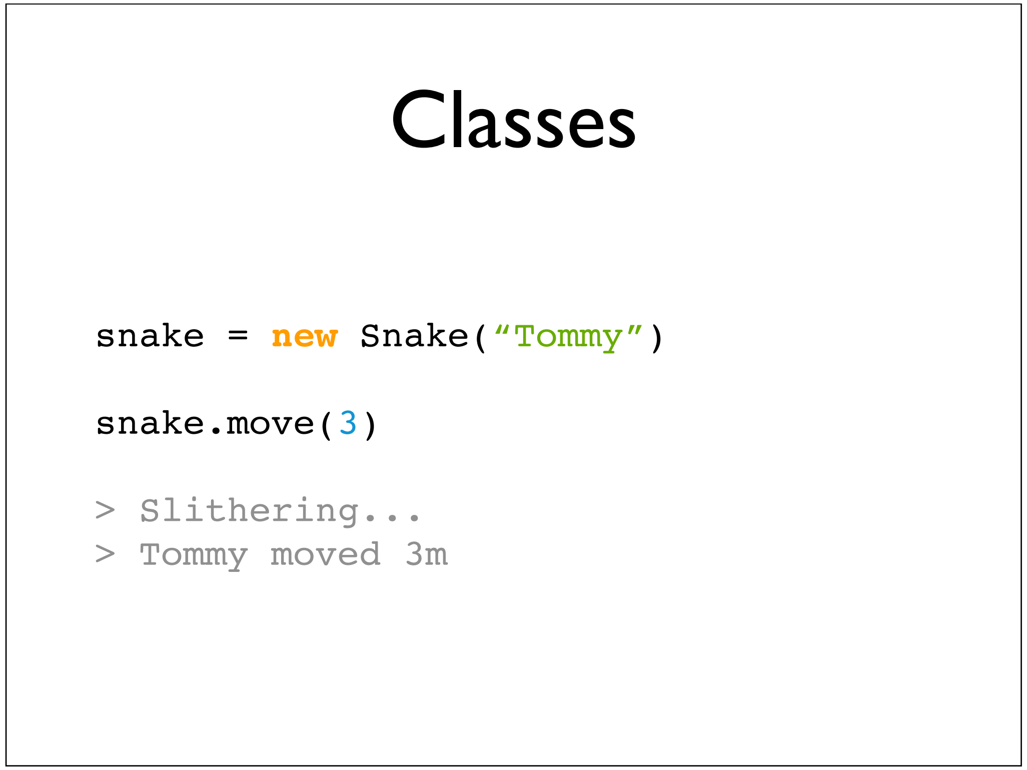
inheritance

Classes

```
class Snake extends Animal

  move: (m) ->
    console.log “Slithering...”
    super m
```

and super



Classes

```
snake = new Snake(“Tommy”)

snake.move(3)

> Slithering...
> Tommy moved 3m
```

inheritance

Thanks!

Thanks!

http://github.com/foca

@godfoca

cubox

Questions?

http://github.com/foca

@godfoca

cubox