A Sip of

CoffeeScript

Follow along! http://172.30.2.43:8080

(Firefox recommended)

Joe Barnes

@joescii

Powered by Lift

Sip of
cript

ttp://172.30.2.43:8080
recommended)

Joe Barnes
@joescii

- Primarily Java EE from 2004-2013
- Began CoffeeScript in August 2013
- Chose Scala/CoffeeScript at Mentor Graphics late 2013

A Sip of

What is CoffeeScript?

Follow along! http://172.30.2.43:8080

(Firefox recommended)

Powered by Lift

Joe Barnes

@joescii

- Primarily Java EE from 2004-2
- Began CoffeeScript in August
- Chose Scala/CoffeeScript at M
  Graphics late 2013

CoffeeScript is a little language that compiles into JavaScript.

Compiles one-to-one into the equivalent JS, and there is no interpretation at runtime

You can use any existing JavaScript library seamlessly from CoffeeScript

CoffeeScript

JS

A Sip of

Follow along! http://172.30.2.43:8080
(Firefox recommended)

CoffeeScript

Joe Barnes
@joescii

- Primarily Java EE from 2004-2013
- Began CoffeeScript in August 2013
- Chose Scala/CoffeeScript at Mentor Graphics late 2013

It's just javascript!

This is a big deal if you've used to languages where don't require semi-colons!

Ok, so… why bother?

Javascript is evil.

And I ain't the first to say it.

Ok, so... why bother?

Take for instance, semi-colons...
which are semi-required.

Flip

coffee

```coffee
# say hello!

alert('hello world!')

alert('can use semis');

alert('but not required')
```

javascript

```javascript
alert('hello world!');

alert('can use semis');

alert('but not required');

```

This is a big deal if you're used to languages which don't require semi-colons!

In fact, much syntax is optional or unneeded entirely

Flip

coffee

```coffee
# wrap me up!!

alert('all cozy!')
```

javascript

```javascript
(function() {
  alert('all cozy!');

}).call(this);

```

But we'll keep that turned off for your viewing pleasure

Flip coffee

```coffee
# Declaring vars

num = 42

if num < 0
  negative = true
else if num > 0
  positive = true
else
  zero = true

# To be equal...
# or not to be equal?

if num % 2 == 0
  even = true
else if num % 2 is 1
  odd = true

# Keywords escape!

obj =
  delete: "delete"
  with: "with"
  void: "void"
```

javascript

```javascript
var even, negative, num, obj, odd,

num = 42;

if (num < 0) {
  negative = true;
} else if (num > 0) {
  positive = true;
} else {
  zero = true;
}

if (num % 2 === 0) {
  even = true;
} else if (num % 2 === 1) {
  odd = true;
}

obj = {
  "delete": "delete",
  "with": "with",
  "void": "void"
};

```

Flip

coffee

```coffee
# Existence

alert 'a global!' if global?

# Chaining null sinks

stuff =
  window.fn?().global?.stuff

# undefined keyword

keyword = undefined

```

javascript

```javascript
var keyword, stuff, _ref;

if (typeof global !== "undefined"
  alert('a global!');
}

stuff = typeof window.fn === "func

keyword = void 0;

```

Flip coffee

```coffee
# Lean function syntax

square = (n) -> n * n

# No args? No problem!
describe 'square', ->
  it 'should work', ->
    expect(square 2).toBe 4
    expect(square -3).toBe 9

# Remember var safety?

arr = [1, 2, 3]
swap = (a) ->
  temp = a[0]
  a[0] = a[1]
  a[1] = temp

# Splat!

concat = (arr, as...) ->
  for a in as
    arr.push(a)
  return arr

```

javascript

```javascript
var arr, concat, square, swap,
  __slice = [].slice;

square = function(n) {
  return n * n;
};

describe('square', function() {
  return it('should work', functio
    expect(square(2)).toBe(4);
    return expect(square(-3)).toBe
  });
});

arr = [1, 2, 3];

swap = function(a) {
  var temp;
  temp = a[0];
  a[0] = a[1];
  return a[1] = temp;
};

concat = function() {
  var a, arr, as, _i, _len;
  arr = arguments[0], as = 2 <= ar
  for ( i = 0 , len = as.length;
```

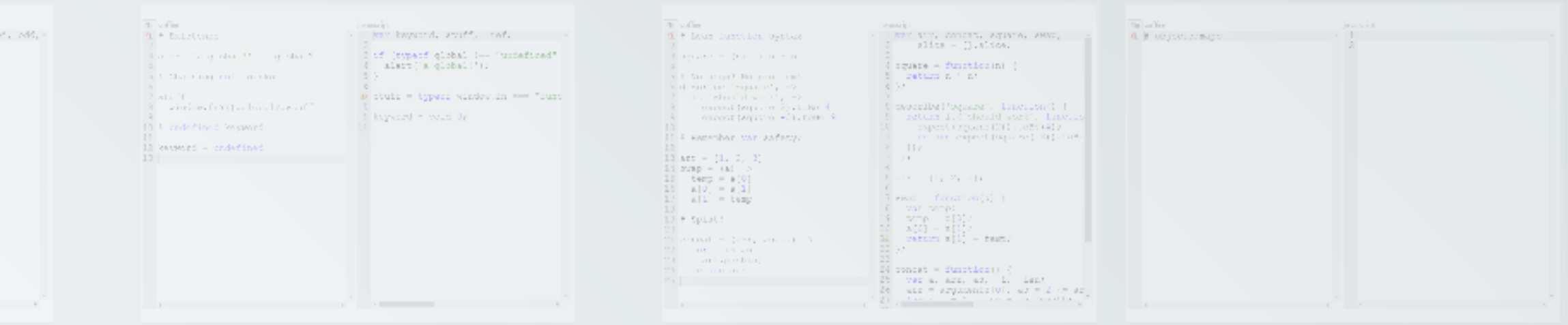
```
23      arr.push(a)
24    return arr
25
```

```
24 concat = function() {
25   var a, arr, as, _i, _len;
26   arr = arguments[0], as = 2 <= ar
27   for ( i = 0   len = as.length;
```

`return` is always implied, because a function body is an *expression*.

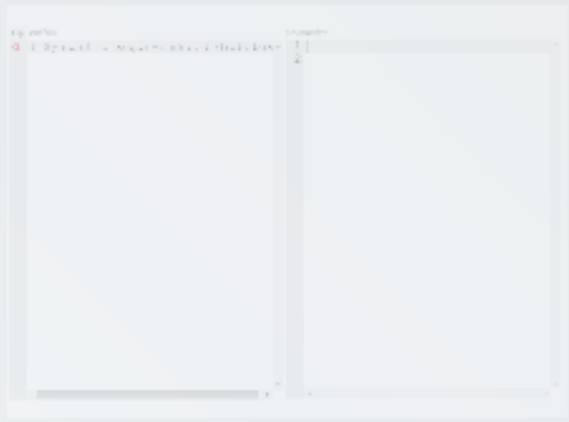
End digression

More cool stuff, anyone??



DIGRESSION

`return` is always implied, because a function body is an *expression*.



`return` is always implied, because a function body is an *expression*.

What is an expression?

`return` is always implied, because a function body is an *expression*.

What is a statement?

What is an expression?

`return` is always implied, because a function body is an *expression*.

How vs. What

Flip

coffee

```coffee
# if/else statement

a = 42; b = 31

if a < b
  min = a
else
  min = b

# if/else expression

max = if a > b then a else b
```

javascript

```javascript
var a, b, max, min;

a = 42;

b = 31;

if (a < b) {
  min = a;
} else {
  min = b;
}

max = a > b ? a : b;
```

Flip coffee 1 spec, 0 failures in 0.02s (see results)

```coffee
# Express all the things!

sign = (x) ->
  if x < 0
    "negative"
  else if x > 0
    "positive"
  else
    "zero"


describe 'sign', ->
  it 'should work', ->
    expect(sign(-5))
    .toBe "negative"
    expect(sign(0))
    .toBe "zero"
    expect(sign(10))
    .toBe "positive"
```

javascript

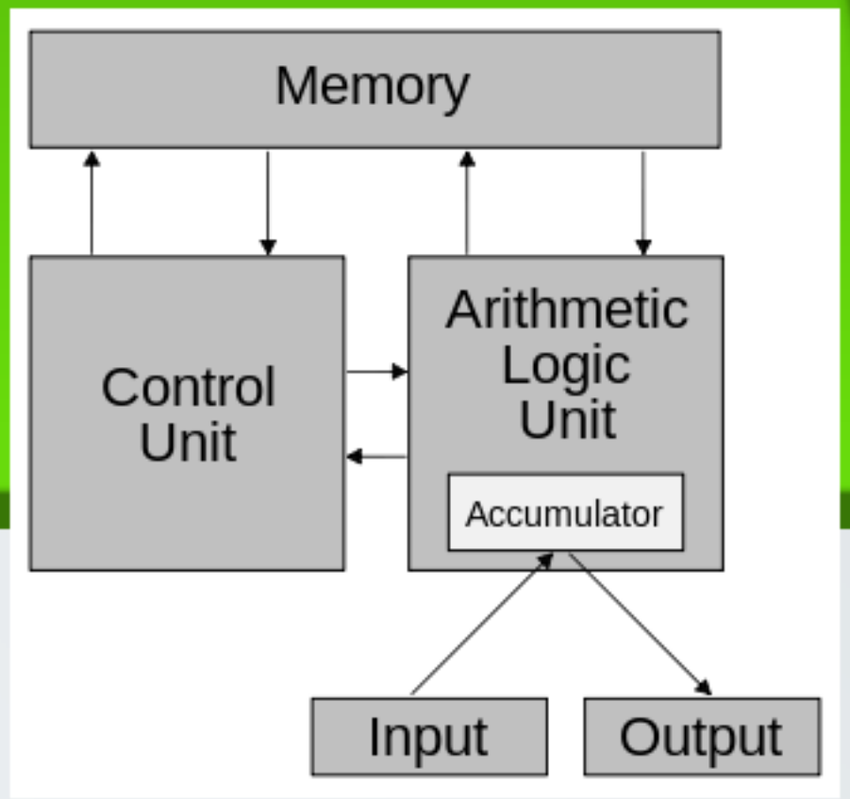
```javascript
var sign;

sign = function(x) {
  if (x < 0) {
    return "negative";
  } else if (x > 0) {
    return "positive";
  } else {
    return "zero";
  }
};

describe('sign', function() {
  return it('should work', functio
    expect(sign(-5)).toBe("negativ
    expect(sign(0)).toBe("zero");
    return expect(sign(10)).toBe("
  });
});

```

Flip coffee 1 spec, 0 failures in 0.019s (see results)

```coffee
ns = [-3..4]
square = (n) -> n * n

# for the win!

squares = (square n for n in ns when n > 0)

describe 'positive squares', ->
  it 'should work', ->
    expect(squares)
    .toEqual([1,4,9,16])
```

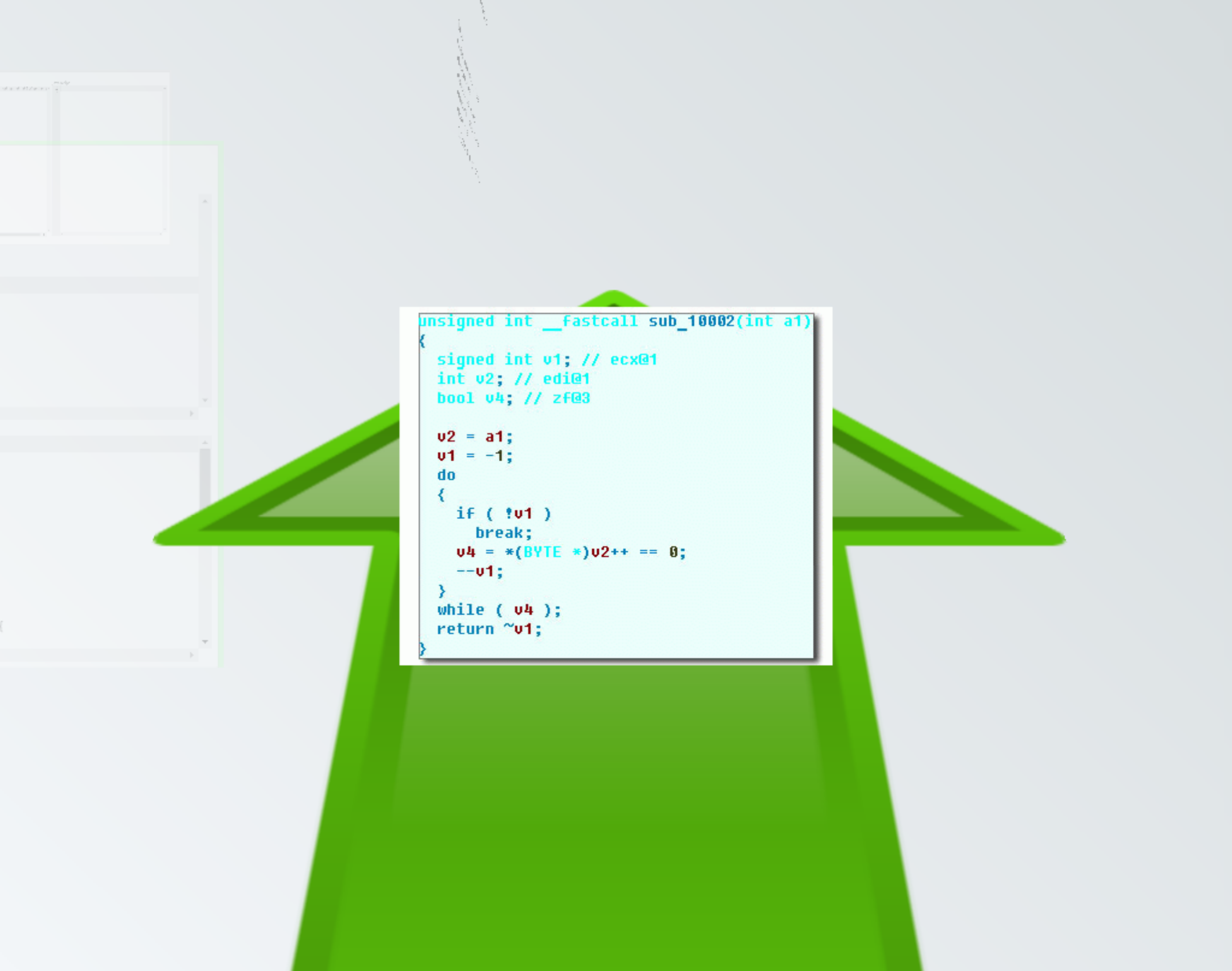
javascript

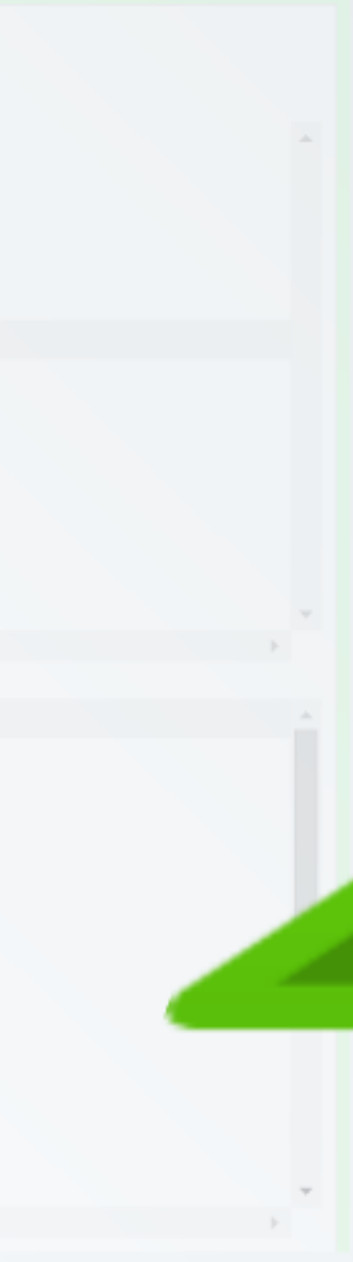
```javascript
var n, ns, square, squares;

ns = [-3, -2, -1, 0, 1, 2, 3, 4];

square = function(n) {
  return n * n;
};

squares = (function() {
  var _i, _len, _results;
  _results = [];
  for (_i = 0, _len = ns.length; _i < _len; _i++) {
    n = ns[ i];
```


Memory
Control Unit
Arithmetic Logic Unit
Accumulator
Input
Output


1001001011011010101
0101100101001110100
1000101110000110110
1010000001010110101
1100101010110000101
1011001000110001000

```
simple_loop:
# parameter 1: %rdi
..B1.1:                         # Preds ..B1.0
..___tag_value_simple_loop.1:   #2.1
        xorl      %eax, %eax    #3.19
        xorl      %edx, %edx    #5.8
        testq     %rdi, %rdi    #5.16
        jle       ..B1.5        # Prob 10% #5.16
                                #LOE rax rdx rbx rbp rdi r12 r13 r14 r15
..B1.3:                         # Preds ..B1.1 ..B1.3
        addq      %rdx, %rax    #6.5
        addq      $1, %rdx      #5.19
        cmpq      %rdi, %rdx    #5.16
        jl        ..B1.3        # Prob 82%   #5.16
..B1.5:                         # Preds ..B1.3 ..B1.1
        ret                     #8.10
        .align    2,0x90
```


unsigned int __fastcall sub_10002(int a1)
{
signed int v1; // ecx@1
int v2; // edi@1
bool v4; // zf@3
v2 = a1;
v1 = -1;
do
{
if (!v1)
break;
v4 = *(BYTE *)v2++ == 0;
--v1;
}
while (v4);
return ~v1;
}



```cpp
#include "maxcpp5.h"

class Example : public MaxCpp5<Example> {
public:
    Example(t_symbol * sym, long ac, t_atom * av) {
        setupIO(2, 2); // inlets / outlets
    }
    ~Example() {}
    void bang(long inlet) {
        outlet_bang(m_outlet[0]);
    }
    void test(long inlet, t_symbol * s, long ac, t_atom * av) {
        outlet_anything(m_outlet[1], gensym("test"), ac, av);
    }
};

extern "C" int main(void) {
    // create a class with the given name:
    Example::makeMaxClass("example");
    REGISTER_METHOD(Example, bang);
    REGISTER_METHOD_GIMME(Example, test);
}
```

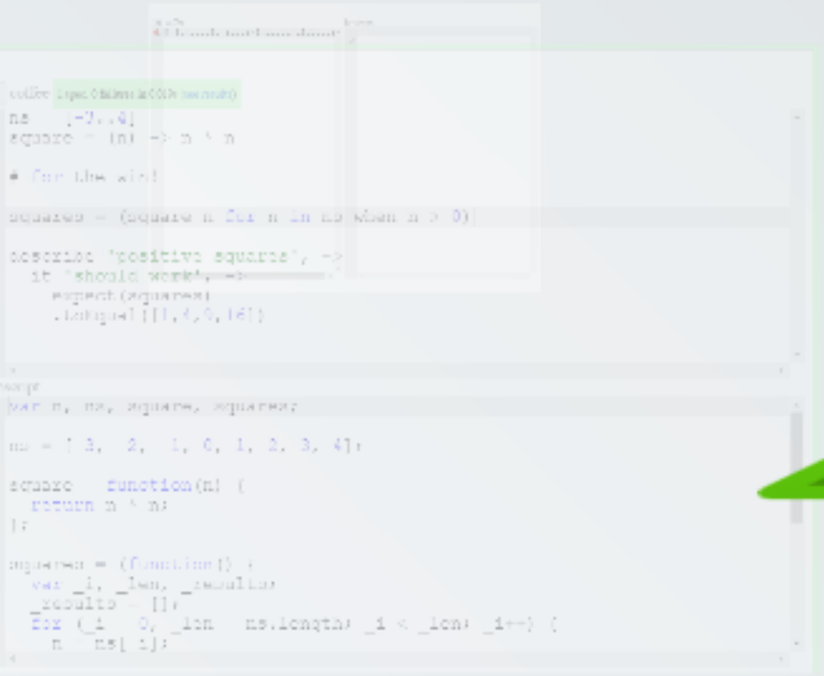
```java
/* Block comment */
import java.util.Date;
/**
 * Doc comment here for SomeClass
 * @version 1.0
 */
public class SomeClass { // some comment
  private String field = "Hello World";
  private double unusedField = 12345.67890;
  private UnknownType anotherString = "AnotherString";
  public SomeClass() {
   //TODO: something
   int localVar = "IntelliJ"; // Error, incompatible types
   System.out.println(anotherString + field + localVar);
   long time = Date.parse("1.2.3"); // Method is deprecated
  }
}
```

Thank you.

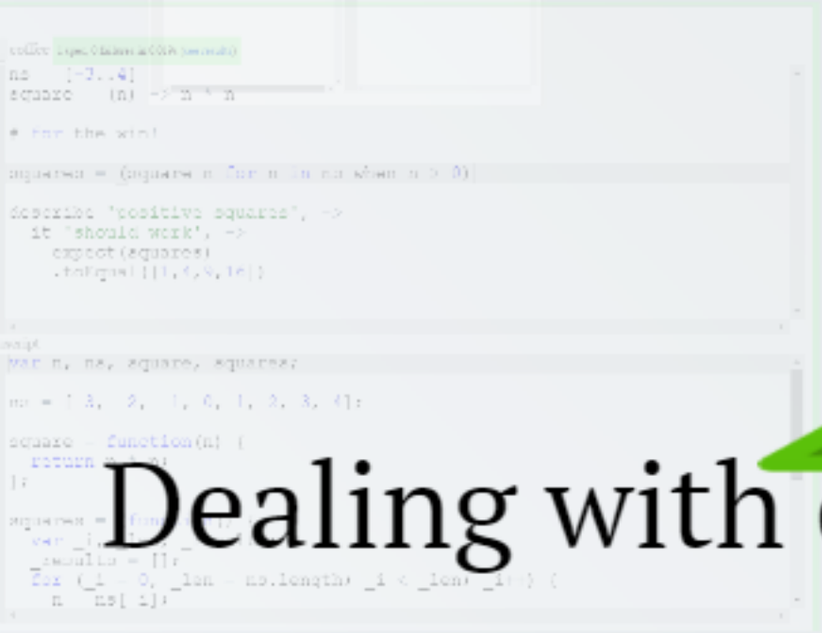
Questions, comments, etc.:

Please fill out the feedback

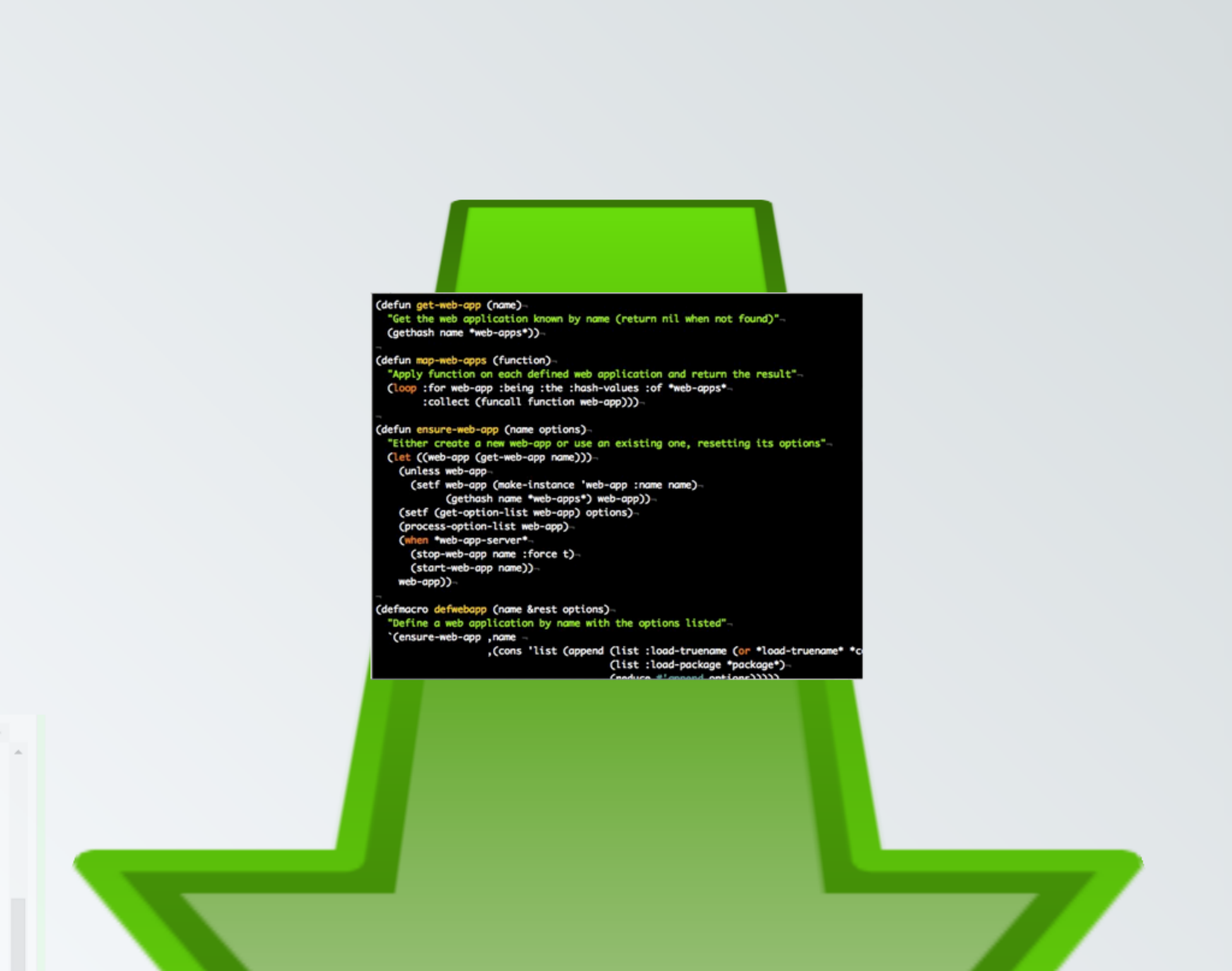
Presentation ends



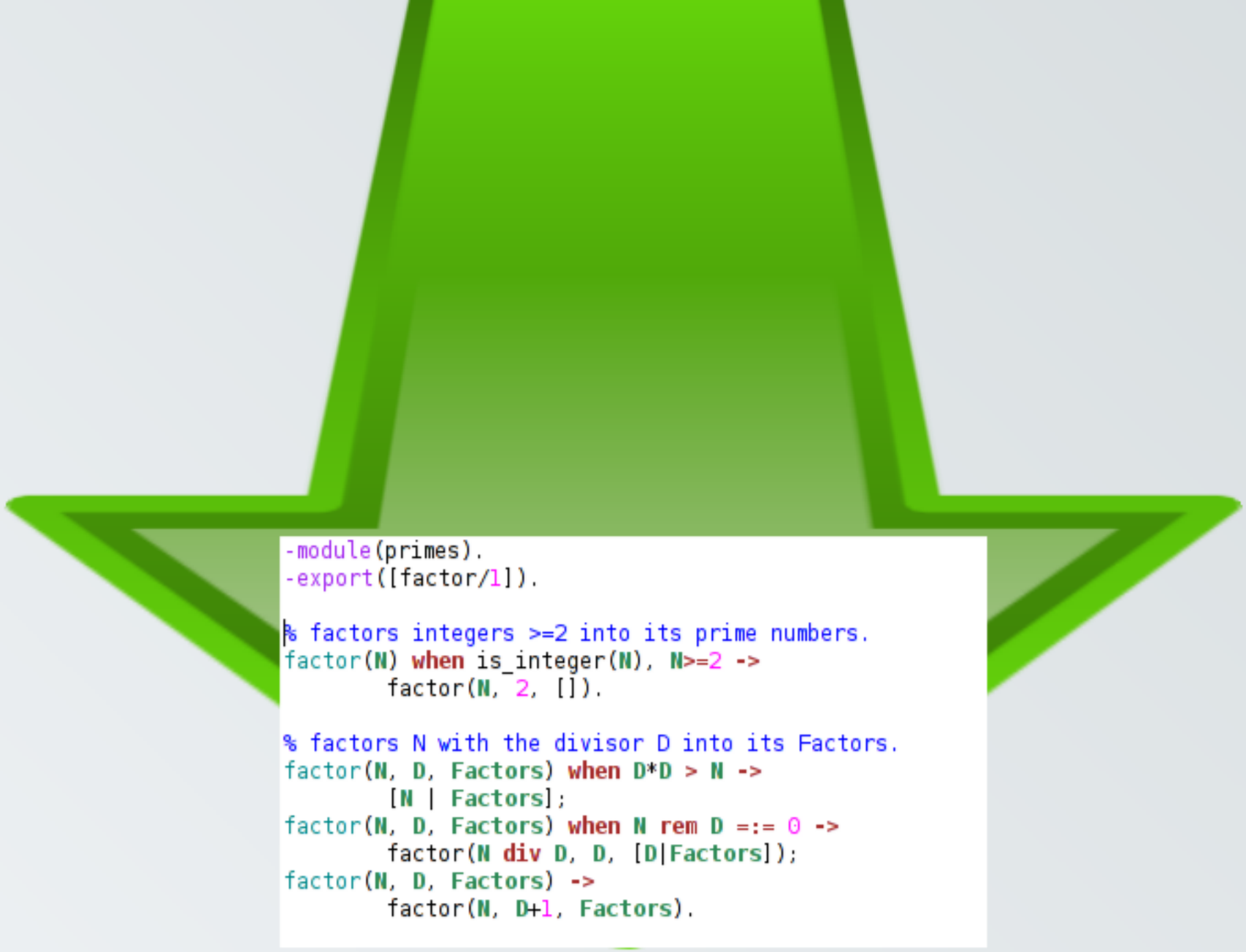
History of imperative programming in a nutshell...



Dealing with difficulty by increasing abstraction


(defun get-web-app (name)
"Get the web application known by name (return nil when not found)"
(gethash name *web-apps*))
(defun map-web-apps (function)
"Apply function on each defined web application and return the result"
(loop :for web-app :being :the :hash-values :of *web-apps*
:collect (funcall function web-app)))
(defun ensure-web-app (name options)
"Either create a new web-app or use an existing one, resetting its options"
(let ((web-app (get-web-app name)))
(unless web-app
(setf web-app (make-instance 'web-app :name name)
(gethash name *web-apps*) web-app))
(setf (get-option-list web-app) options)
(process-option-list web-app)
(when *web-app-server*
(stop-web-app name :force t)
(start-web-app name))
web-app))
(defmacro defwebapp (name &rest options)
"Define a web application by name with the options listed"
`(ensure-web-app ,name
,(cons 'list (append (list :load-truename (or *load-truename* *c
(list :load-package *package*)

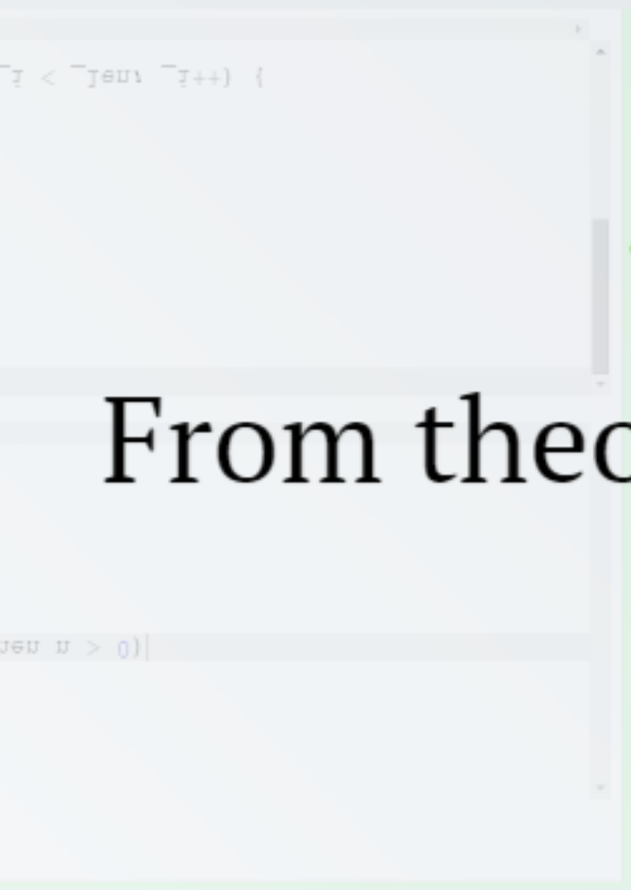
```haskell
-- Type annotation (optional)
fib :: Int -> Integer

-- With self-referencing data
fib n = fibs !! n
        where fibs = 0 : scanl (+) 1 fibs
        -- 0,1,1,2,3,5,...

-- Same, coded directly
fib n = fibs !! n
        where fibs = 0 : 1 : next fibs
              next (a : t@(b:_)) = (a+b) : next t

-- Similar idea, using zipWith
fib n = fibs !! n
        where fibs = 0 : 1 : zipWith (+) fibs (tail fibs)

-- Using a generator function
fib n = fibs (0,1) !! n
        where fibs (a,b) = a : fibs (b,a+b)
```



```erlang
-module(primes).
-export([factor/1]).

% factors integers >=2 into its prime numbers.
factor(N) when is_integer(N), N>=2 ->
        factor(N, 2, []).

% factors N with the divisor D into its Factors.
factor(N, D, Factors) when D*D > N ->
        [N | Factors];
factor(N, D, Factors) when N rem D =:= 0 ->
        factor(N div D, D, [D|Factors]);
factor(N, D, Factors) ->
        factor(N, D+1, Factors).
```

```scala
package ListMethods;

object ListMethods {
  def main(args: Array[String]) {
    /// listcat.scala
    //  Scala List - prepend operator
    val oneTwo = List(1, 2)
    val threeFour = List(3, 4)
    val oneTwoThreeFour = oneTwo ::: threeFour  // prepend
    println(oneTwo + " and " + threeFour + " were not mutated.")
    println("This " + oneTwoThreeFour + " is a new List.")
```

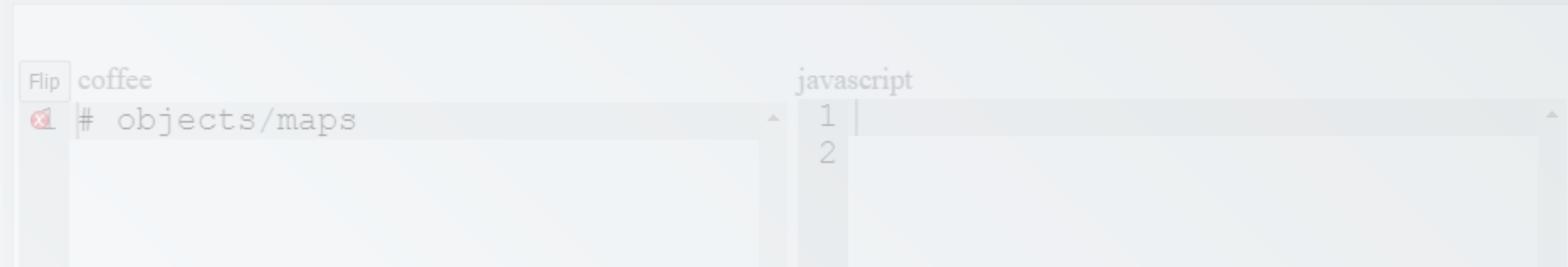
History of functional programming in a nutshell...



From theory down to practice

End digression

More cool stuff, anyone??



Flip coffee

```coffee
familia =
  madre:
    nombre: 'Tammy'
    edad: 54
  padre:
    nombre: 'Mike'
    edad: 63

espanol = uno:1, dos:2, tres:3

msg = for esp, num of espanol when num % 2 isnt 0
  "#{num} is #{esp}"

```

javascript

```javascript
var esp, espanol, familia, msg, num;

familia = {
  madre: {
    nombre: 'Tammy',
    edad: 54
  },
  padre: {
    nombre: 'Mike',
    edad: 63
  }
};

```

Flip coffee

```coffee
# destructuring

joe =
  twitter: 'joescii'
  blog: 'http://proseand.co.nz'
  address: ['Athens', 'AL']

{address: [city, state]} = joe

```

javascript

```javascript
var city, joe, state, _ref;

joe = {
  twitter: 'joescii',
  blog: 'http://proseand.co.nz',
  address: ['Athens', 'AL']
};

_ref = joe.address, city = _ref[0], state = _ref[1];

```

Flip

coffee

```coffee
# switches

lunch = switch day
  when "Mon","Tue","Thr" then "sal
  when "Wed" then "wings"
  when "Fri" then "happy hour"
  else "sandwich"

```

javascript

```javascript
var lunch;

lunch = (function() {
  switch (day) {
    case "Mon":
    case "Tue":
    case "Thr":
      return "salad";
    case "Wed":
      return "wings";
    case "Fri":
      return "happy hour";
    default:
      return "sandwich";
  }
})();

```

Flip coffee

```coffee
# Clean OOP

class Person
  constructor: (@name) ->

class Doctor extends Person
  work: ->
    alert 'I cutz you up!'
  play: ->
    alert 'Golf swing'

class Developer extends Person
  work: ->
    alert 'I rite teh codez!'
  play: ->
    @work()
```

javascript

```javascript
var Developer, Doctor, Person,
  __hasProp = {}.hasOwnProperty,
  __extends = function(child, pare

Person = (function() {
  function Person(name) {
    this.name = name;
  }

  return Person;

})();

Doctor = (function(_super) {
  __extends(Doctor, _super);

  function Doctor() {
    return Doctor.__super__.constr
  }

  Doctor.prototype.work = function
    return alert('I cutz you up!')
  };

  Doctor.prototype.play = function
    return alert('Golf swing');
  };
```

Flip coffee

```coffee
# The fat arrow!

Widget = (data) ->
  @data = data
  update = (data) =>
    @data = data

  $('.my-div').bind 'click',
    (event) => @data = 'clicked'


# The prototype...

Array::contains = (item) ->
  @indexOf(item) != -1

```

javascript

```javascript
var Widget;

Widget = function(data) {
  var update;
  this.data = data;
  update = (function(_this) {
    return function(data) {
      return _this.data = data;
    };
  })(this);
  return $('.my-div').bind('click'
    return function(event) {
      return _this.data = 'clicked
    };
  })(this));
};

Array.prototype.contains = functio
  return this.indexOf(item) !== -1
};

```

At last! The Silver Bullet! God be praised!

Because it compiles to JavaScript, it cannot solve all of the problems.

Runtime train wrecks like eval, typeof, instanceof, delete, etc are still at your disposal.

Implicit returns don't always play well with existing JavaScript libraries like Angular.

```
19  }
20
21  Doctor.prototype.work = function
22    return alert('I cutz you up!')
23  };
24
25  Doctor.prototype.play = function
26    return alert('Golf swing');
27  };
```

CoffeeScript has its own "bad parts"

Flip

coffee

```coffee
# Syntactic sugar-induced diabetes

f = (a, b) ->
g = (c, d) ->

result = f g 2, 3


obj = k1: "1", k2: "2"

f k1: "1", k2: "2"
```

javascript

```javascript
var f, g, obj, result;

f = function(a, b) {};

g = function(c, d) {};

result = f(g(2, 3));

obj = {
  k1: "1",
  k2: "2"
};

f({
  k1: "1",
  k2: "2"
});

```

So what's the cost?

Add the compiler to your toolchain

(No big deal since you minify already)

But compilation *does* impact debugging...

Recap:

- Avoids common JavaScript pitfalls

Recap:

- Avoids common JavaScript pitfalls
- Less syntactic noise
- Expression-based semantics
- Cannot avoid JS runtime problems
- No silver bullet

Is CoffeeScript right for you?

Thank you!

Questions, comments, concerns?

Please fill out the feedback!

Presentation source