Hack your language with Rascal

Jouke Stoel & Tijs van der Storm
@jstoel / @tvdstorm

JS +  = *Sweeter* JS

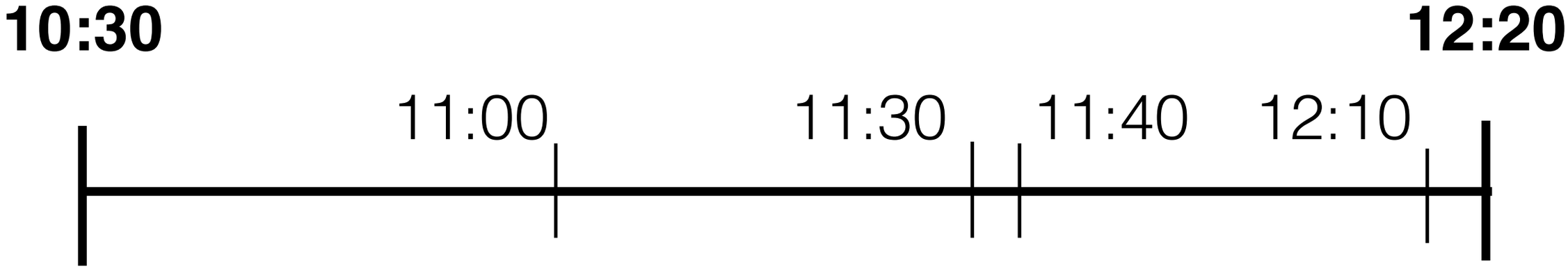
Joy of Coding

CWI
Centrum Wiskunde & Informatica

Universiteit van Amsterdam


STOP
FASTEN
YOUR
SEAT BELTS

Outline of the workshop



10:30 - 11:00 Introduction

11:00 - 11:30 Hands-on part 1

11:30 - 11:40 Discussing the answers

11:40 - 12:10 Hands-on part 2

12:10 - 12:20 Wrap-up

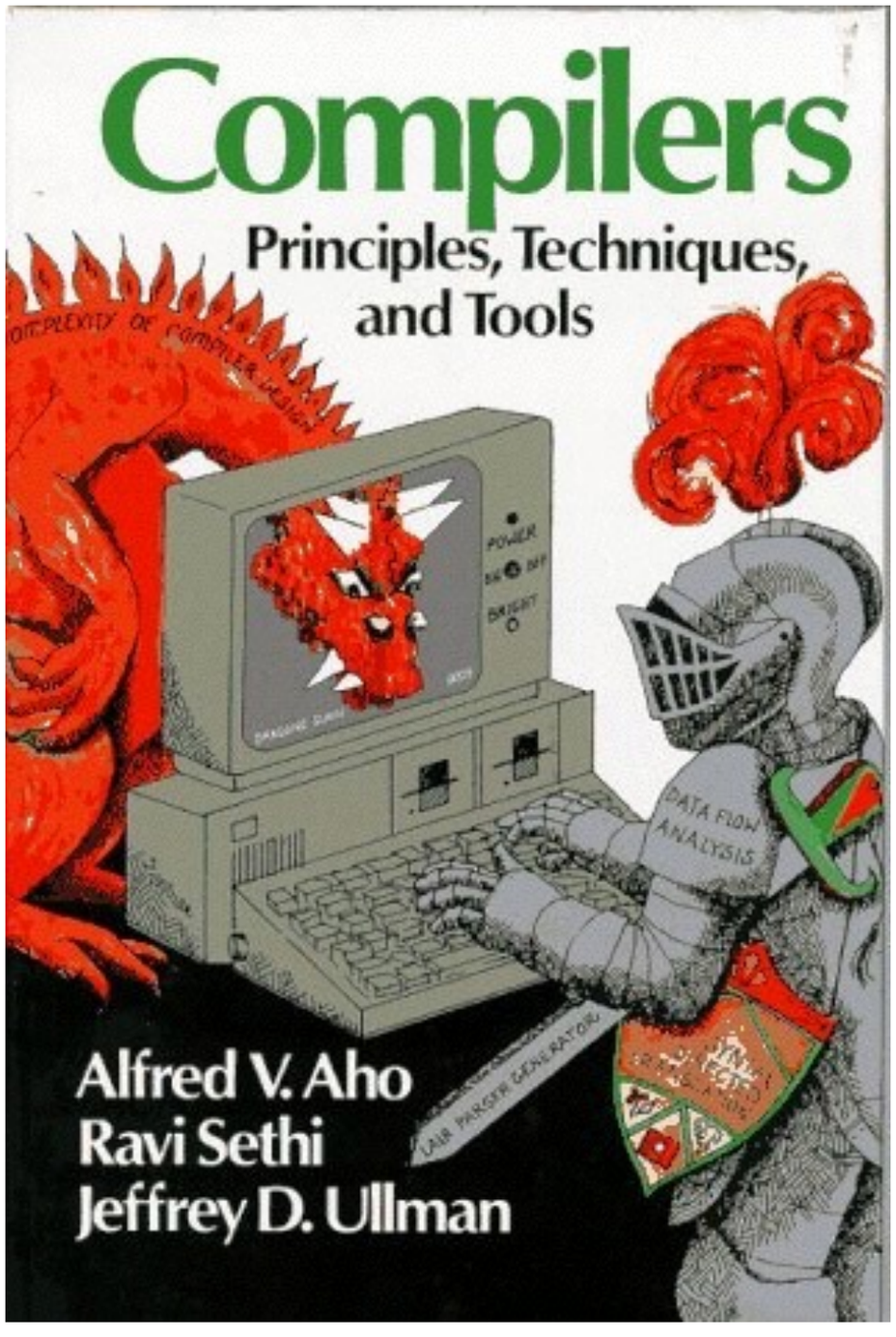
Why hack your language?

What do we mean?

Syntactic abstraction

functions
objects
data types
libraries
frameworks
macros
…
language constructs

How to do it?



Build a compiler!

Too hard!

How to do it?

Use alien technology!
(aka Lisp)

Too esoteric!

How to do it?

Use a language workbench!

Rascal

http://www.rascal-mpl.org

CWI
Centrum Wiskunde & Informatica

Sneak preview

Select query

```
var q = select FirstName, LastName
        from myList
        where FirstName === "Chris";
```

HAML

```
var doc = %html {
  %head  %title "Hello Joy of Coding!";
  %body #main {
    %h1 "Hello Joy of Coding!";
    %ul for (var n in names)
      %li { "Hello "; n; "!"; 3; }
  }
};
```

http://haml.info/

State machines

```
var doors = statemachine {
    state closed {
        console.log("Door is closed");
        on open goto opened;
    }
    state opened {
        console.log("Door is opened");
        on close goto closed;
    }
};
```

Rascal in a nutshell

- A **meta programming language** for **source code analysis and transformation**

- Java like syntax, but functional language
  - Immutable data, higher-order, algebraic data types etc.

- Powerful primitives for parsing, pattern matching, comprehensions, relation calculus, tree traversal

- Integration with Eclipse IDE

What is desugaring?

Source to source transformation

```
var q = select FirstName, LastName
        from myList
        where FirstName === "Chris";
```

```
var q = JSLINQ(myList)
  .Where(function(item) {
     return item.FirstName === "Chris";
  })
  .Select(function (item) {
     return {FirstName: item.FirstName
             ,LastName: item.LastName};
  });
```

Source to source transformation

```
var q = select FirstName, LastName
        from myList
        where FirstName === "Chris";
```

Extended syntax

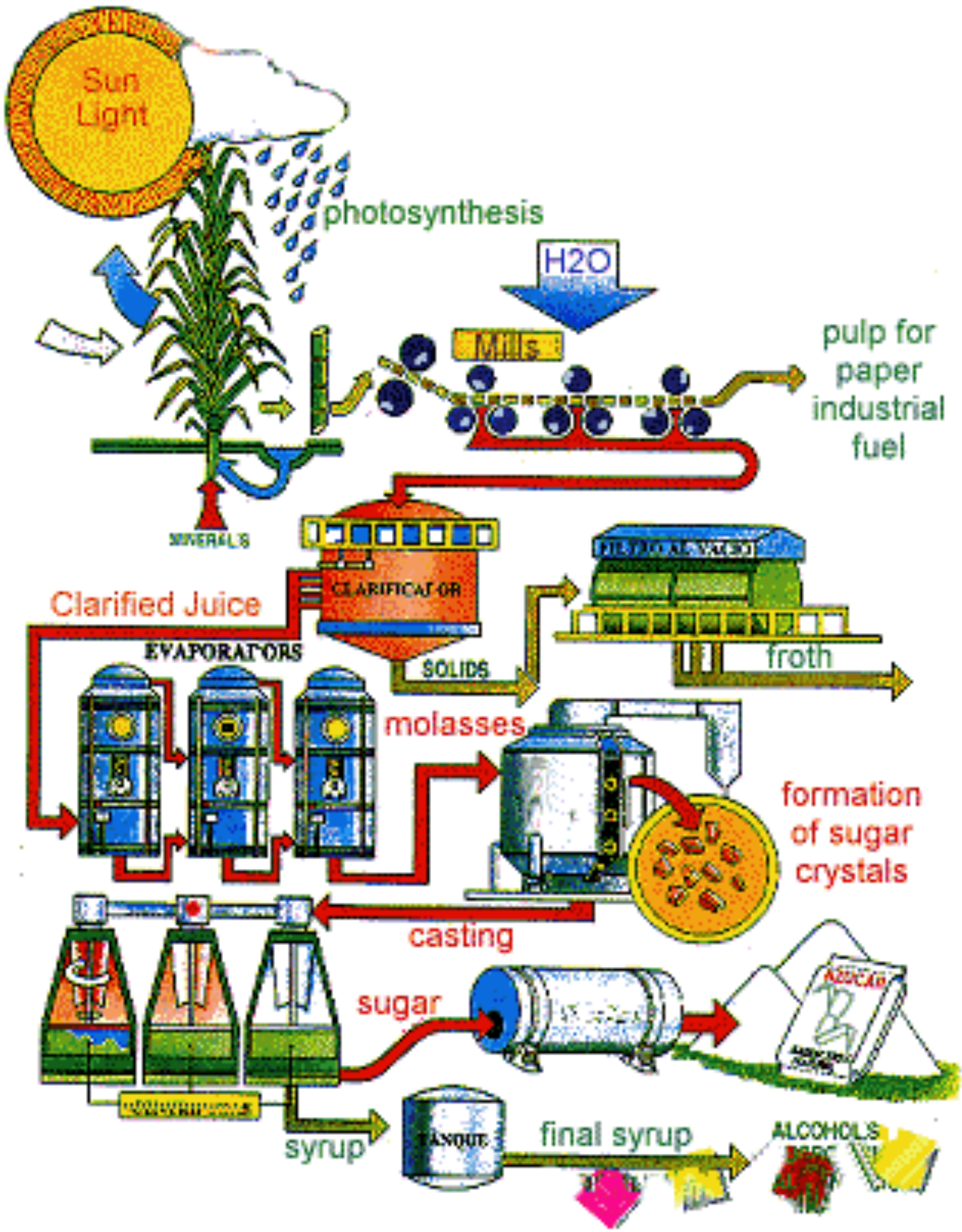
```
var q = JSLINQ(myList)
  .Where(function(item) {
     return item.FirstName === "Chris";
  })
  .Select(function (item) {
     return {FirstName: item.FirstName
             ,LastName: item.LastName};
  });
```

Source to source transformation

```
var q = select FirstName, LastName
        from myList
        where FirstName === "Chris";
```

Base syntax

```
var q = JSLINQ(myList)
  .Where(function(item) {
     return item.FirstName === "Chris";
  })
  .Select(function (item) {
     return {FirstName: item.FirstName
             ,LastName: item.LastName};
  });
```



How to desugar?

How to define a desugaring?

- Extend the **grammar** of the **base language**
- Transform **extended syntax** to **base syntax**

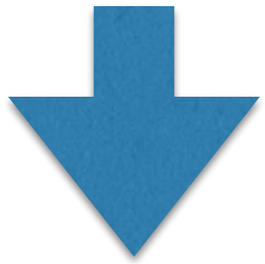
Example

debug statement

“print something if debug == true”

Debug statement

```
debug "Hello";
```



```
if (DEBUG_FLAG)
   console.log("Hello");
```

Original JS grammar

```
start syntax Source
  = source: Statement*
  ;

syntax Statement
  = varDecl: VarDecl
  | empty: ";"
  | block: "{" Statement* "}"
  | expression: Expression!function ";"

  // Block level things
  | function: Function
  | ifThen: "if" "(" Expression cond ")" Statement () !>> "else"
  | ifThenElse: "if" "(" Expression cond ")" Statement "else" Statement
  | doWhile: "do" Statement "while" "(" Expression cond ")" ";"
  | whileDo: "while" "(" Expression cond ")" Statement
  | forIn: "for" "(" Expression var "in" Expression obj ")" Statement
  | forInDeclaration: "for" "(" "var" Id "in" Expression obj ")" Statement
  | with: "with" "(" Expression scope ")" Statement

. . .
```

Extending JS grammar

```
syntax Statement
  = varDecl: VarDecl
  | empty: ";"
  | block: "{" Statement* "}"
  | expression: Expression!function ";"
```

Extending JS grammar

```
syntax Statement
  = varDecl: VarDecl
  | empty: ";"
  | block: "{" Statement* "}"
  | expression: Expression!function ";"
```

+

```
syntax Statement
  = "debug" Expression ";"
```

Extending JS grammar

```
syntax Statement
  = varDecl: VarDecl
  | empty: ";"
  | block: "{" Statement* "}"
  | expression: Expression!function ";"
```

Base grammar

+

```
syntax Statement
  = "debug" Expression ";"
```

Extending JS grammar

```
syntax Statement
  = varDecl: VarDecl
  | empty: ";"
  | block: "{" Statement* "}"
  | expression: Expression!function ";"
```

+

Extended grammar

```
syntax Statement
  = "debug" Expression ";"
```

Transformation

```
Statement desugar((Statement)`debug <Expression ex>;`) {
  return …;
}
```

Transformation

Desugar function
(automatically called)

```
Statement desugar((Statement)`debug <Expression ex>;`) {
  return …;
}
```

Transformation

Returning a Statement

```
Statement desugar((Statement)`debug <Expression ex>;`) {
  return …;
}
```

Transformation

Matching Statements with *concrete pattern*

```
Statement desugar((Statement)`debug <Expression ex>;`) {
  return …;
}
```

Transformation

```
Statement desugar((Statement)`debug <Expression ex>;`) {
  return …;
}
```

Pattern variable ex of type Expression (*hole*)

Transformation

```
Statement desugar((Statement)`debug <Expression ex>;`) {
  return (Statement) `if (DEBUG_FLAG)
                       console.log(<Expression ex>);`;
}
```

Return a Statement

Transformation

```
Statement desugar((Statement)`debug <Expression ex>;`) {
  return (Statement) `if (DEBUG_FLAG)
                     '   console.log(<Expression ex>);`;
}
```

With a pattern using only base syntax

Transformation

```
Statement desugar((Statement)`debug <Expression ex>;`) {
  return (Statement) `if (DEBUG_FLAG)
                     '   console.log(<Expression ex>);`;
}
```

Containing the original expression

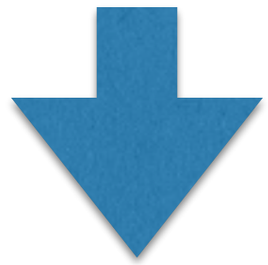
Transformation

```
Statement desugar((Statement)`debug <Expression ex>;`) {
  return (Statement) `if (DEBUG_FLAG)
                     '   console.log(<Expression ex>);`;
}
```

Note the single quote (‘) for multiline patterns

Another example: power

```
var x = 2 ** 3;
```



```
var x = Math.pow(2, 3);
```

Let's add some

to

JS

Setup

- Make sure the Prerequisites are met
- Clone the GIT repo
  - https://github.com/cwi-swat/hack-your-javascript.git
- Open Eclipse Luna
- Open Rascal perspective & import project
- Reload the Language
- Extra help: https://gitter.im/tvdstorm/hack-your-javascript

Exercises - Series 1

- At fields: `@name` ➡ `this.name`
- Twitter: `@("obama")`` ➡ `searchAt("obama")`
- Dont: `dont if (x == 3) print(x);` ➡ `;`
- Todo: `todo “FIXME";` ➡ `console.log("TODO: "+"FIXME");`
- Unless: `unless (x == 0) x;` ➡ `if (!(x == 0)) x;`
- Repeat: `repeat {} until (0);` ➡ `do {} while (!(0));`
- Assert: `assert x != null: "x != null”;` ➡ …

Part 2: names


Changing my name from
"Mom" to "I'mOk!"
has helped my children's
self-sufficiency tremendously!

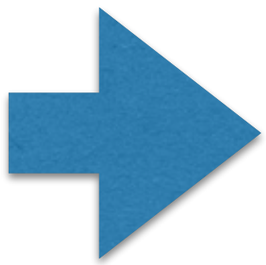
Swap

```
syntax Statement
  = "swap" Id "," Id ";"
  ;

Statement desugar((Statement)`swap <Id x>, <Id y>;`)
  = (Statement) `(function() {
                ‘   var tmp = <Id x>;
                ‘   <Id x> = <Id y>;
                ‘   <Id y> = tmp;
                ‘})();`;
```

Problem 1

```
var x = 1;
var tmp = 2;
swap x, tmp;
```



```
var x = 1;
var tmp = 2;
(function() {
  var tmp = x;
  x = tmp;
  tmp = tmp;
})();
```

Variable capture!

Solved with renaming

```
var x = 1;
var tmp = 2;
swap x, tmp;
```

```
var x = 1;
var tmp = 2;
(function() {
  var tmp$0 = x;
  x = tmp;
  tmp = tmp$0;
})();
```

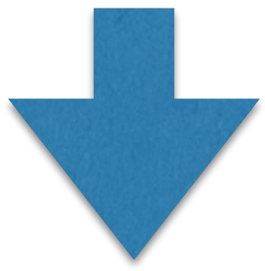
Remember Todo?

```
syntax Statement = "todo" String ";";
```

```
Statement desugar((Statement)`todo <String s>;`)
  = (Statement)`console.log("TODO: " + <String s>);`;
```

Problem 2

```
var console = null;
todo "Fix me";
```

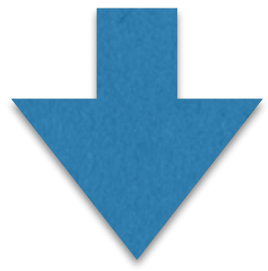


```
var console = null;
console.log("TODO: " + "Fix m
```

Variable capture!

Also solved with renaming

```
var console = null;
todo "Fix me";
```



```
var console$0 = null;
console.log("TODO: " + "Fix me");
```


KEEP
CALM
AND
DESUGAR
ON

Exercises - Series 2

- Swap: `swap x, y;`
- Test: `test 3 * 3 should be 9;`
- Foreach: `foreach (var x in [1,2,3]) print(x);`
- Arrows: `x => this.x + 1`
- Comprehension: `[ i | var i in nums, i % 2 == 0 ]`

Take home message

- Programming languages can be straitjackets
- Syntactic abstraction to the rescue
- Desugaring for extending a language
- Rascal: http://www.rascal-mpl.org
- What's your next extension? :-)