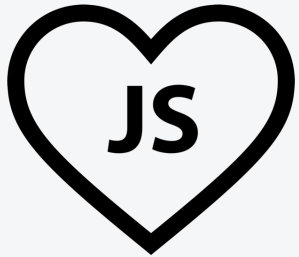
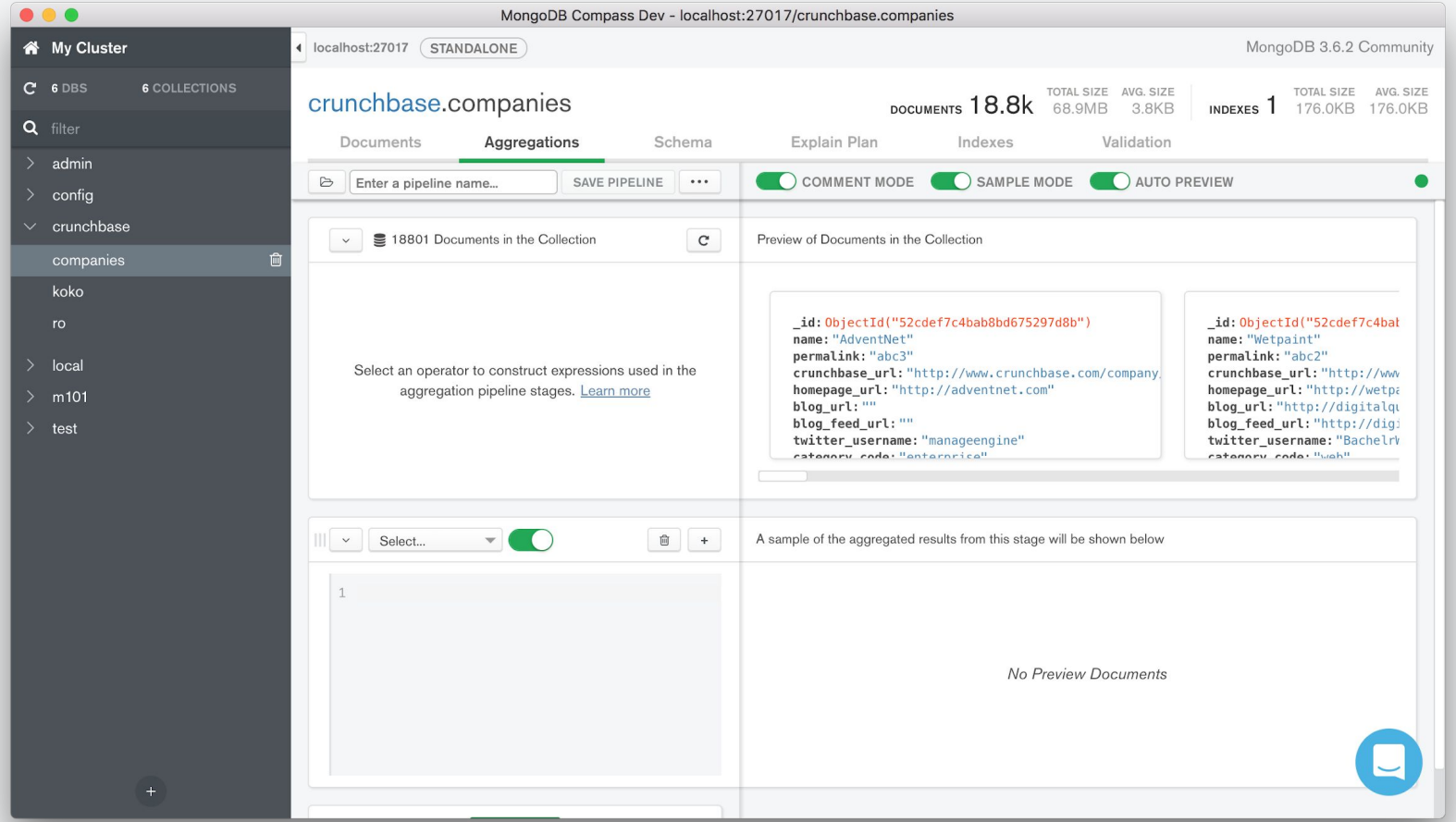
COMPILER

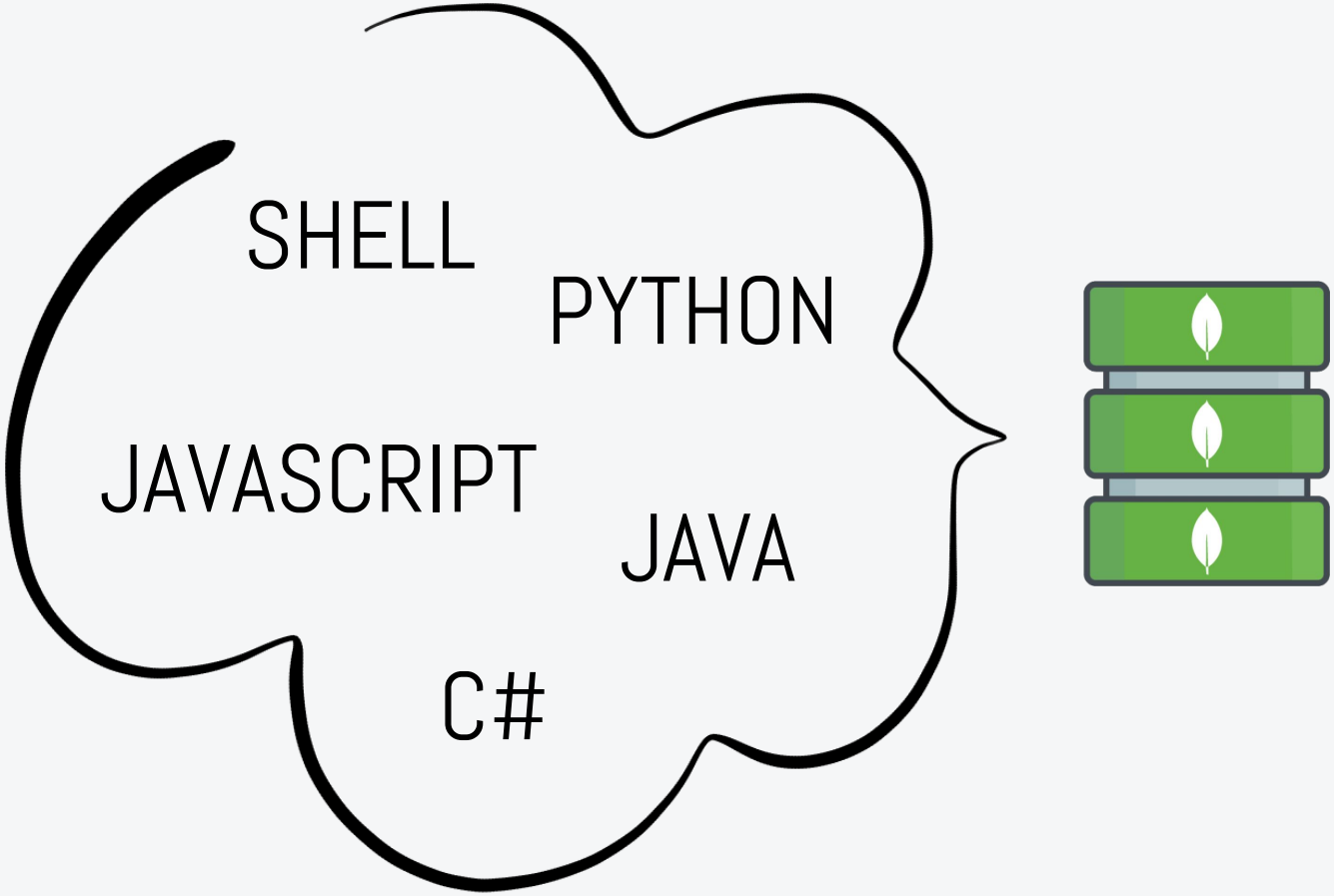
IN JAVASCRIPT USING ANTLR

ALENA KHINEIKA



ALENA KHINEIKA


MongoDB Compass Dev - localhost:27017/crunchbase.companies
My Cluster
6 DBS
6 COLLECTIONS
filter
admin
config
crunchbase
companies
koko
ro
local
m101
test
localhost:27017
STANDALONE
MongoDB 3.6.2 Community
crunchbase.companies
DOCUMENTS 18.8k
TOTAL SIZE 68.9MB
AVG. SIZE 3.8KB
INDEXES 1
TOTAL SIZE 176.0KB
AVG. SIZE 176.0KB
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
Enter a pipeline name...
SAVE PIPELINE
COMMENT MODE
SAMPLE MODE
AUTO PREVIEW
18801 Documents in the Collection
Preview of Documents in the Collection
Select an operator to construct expressions used in the aggregation pipeline stages. Learn more
_id: ObjectId("52cdef7c4bab8bd675297d8b")
name: "AdventNet"
permalink: "abc3"
crunchbase_url: "http://www.crunchbase.com/company
homepage_url: "http://adventnet.com"
blog_url: ""
blog_feed_url: ""
twitter_username: "manageengine"
category_code: "enterprise"
_id: ObjectId("52cdef7c4bat
name: "Wetpaint"
permalink: "abc2"
crunchbase_url: "http://ww
homepage_url: "http://wetp
blog_url: "http://digitalqu
blog_feed_url: "http://digi
twitter_username: "BachelrW
category_code: "web"
Select...
A sample of the aggregated results from this stage will be shown below
1
No Preview Documents


SHELL
PYTHON
JAVASCRIPT
JAVA
C#

SOURCE-TO-SOURCE TRANSFORMATION

SOURCE-TO-SOURCE TRANSFORMATION

- Find existing parsers for specific programming languages

SOURCE-TO-SOURCE TRANSFORMATION

- Find existing parsers for specific programming languages
- Create your own parser

SOURCE-TO-SOURCE TRANSFORMATION

- Find existing parsers for specific programming languages
- Create your own parser
- Use some tools and libraries to generate a parser

JAVASCRIPT PARSERS

ESPRIMA

FALAFEL

UGLIFYJS

JISON

JAVASCRIPT PARSERS

ESPRIMA

FALAFEL

UGLIFYJS

JISON

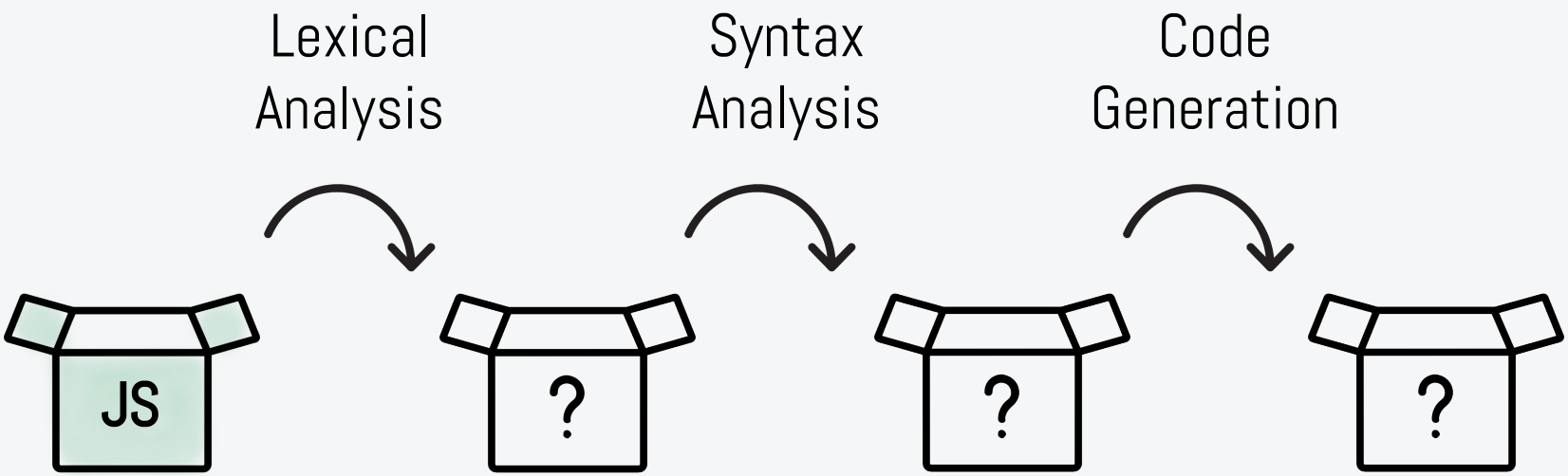
+ BABEL

ANY LANGUAGE TO ANY LANGUAGE

+ BSON DOCUMENTS

HOW TO WRITE
COMPILER FROM SCRATCH?

3 STAGES OF COMPILER DESIGN



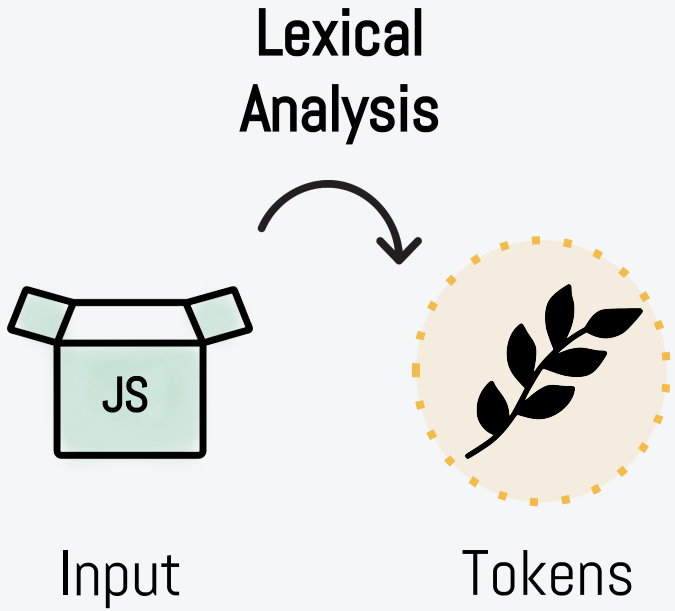
1. LEXICAL ANALYSIS

Raw Code

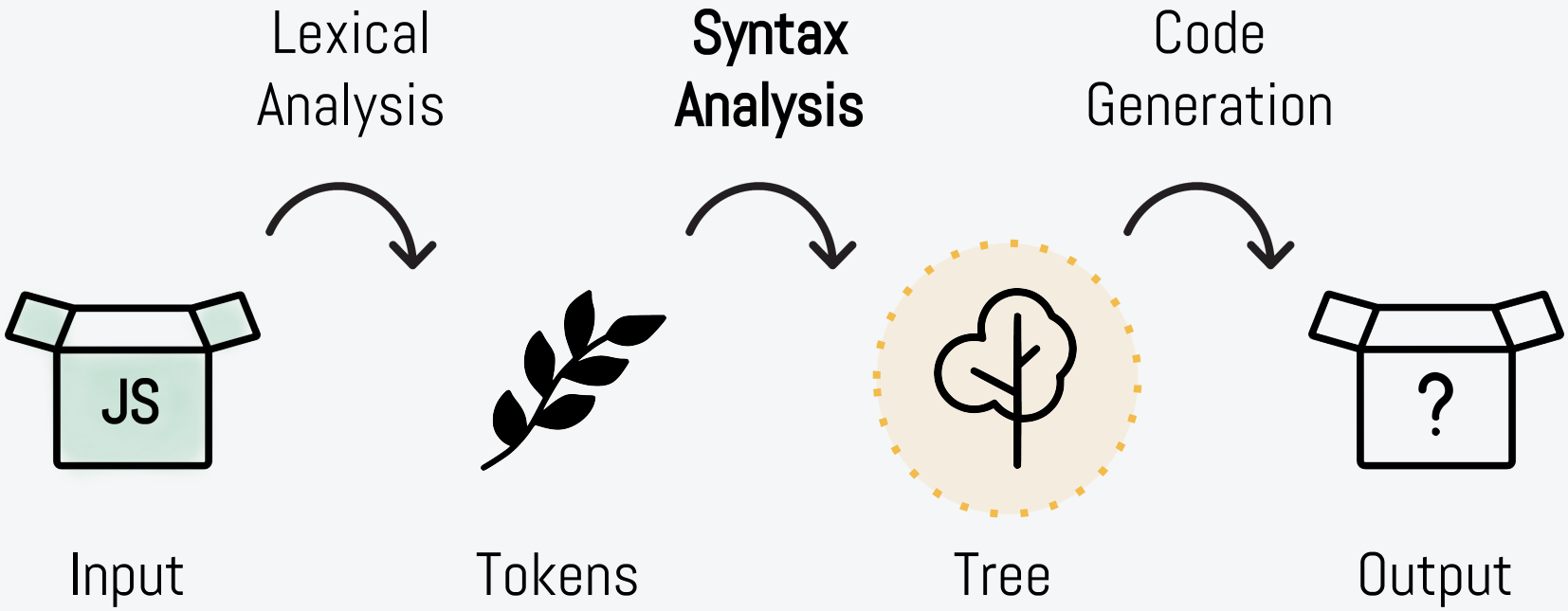
{ x : 1 }

Tokens

OpenBrace	{
Identifier	x
Colon	:
Identifier	1
CloseBrace	}

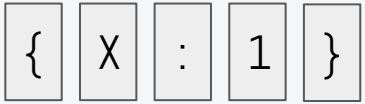


2. SYNTAX ANALYSIS



FROM STRING TO TREE

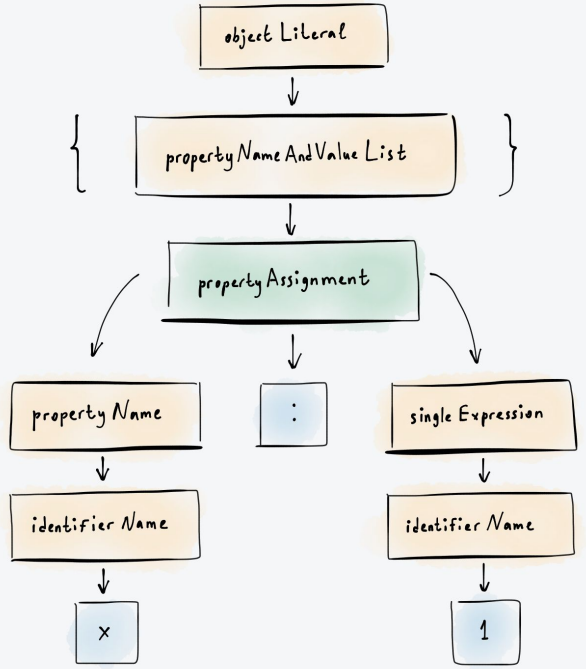
Raw Code



Tokens

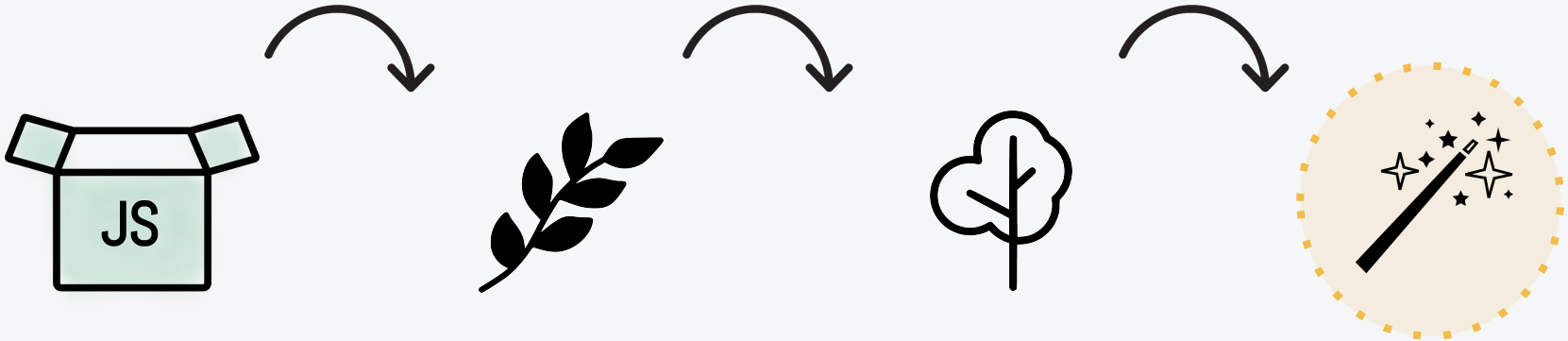
OpenBrace	{
Identifier	x
Colon	:
Identifier	1
CloseBrace	}

Tree Structure



3. CODE GENERATION

Lexical Analysis → Syntax Analysis → **Code Generation**

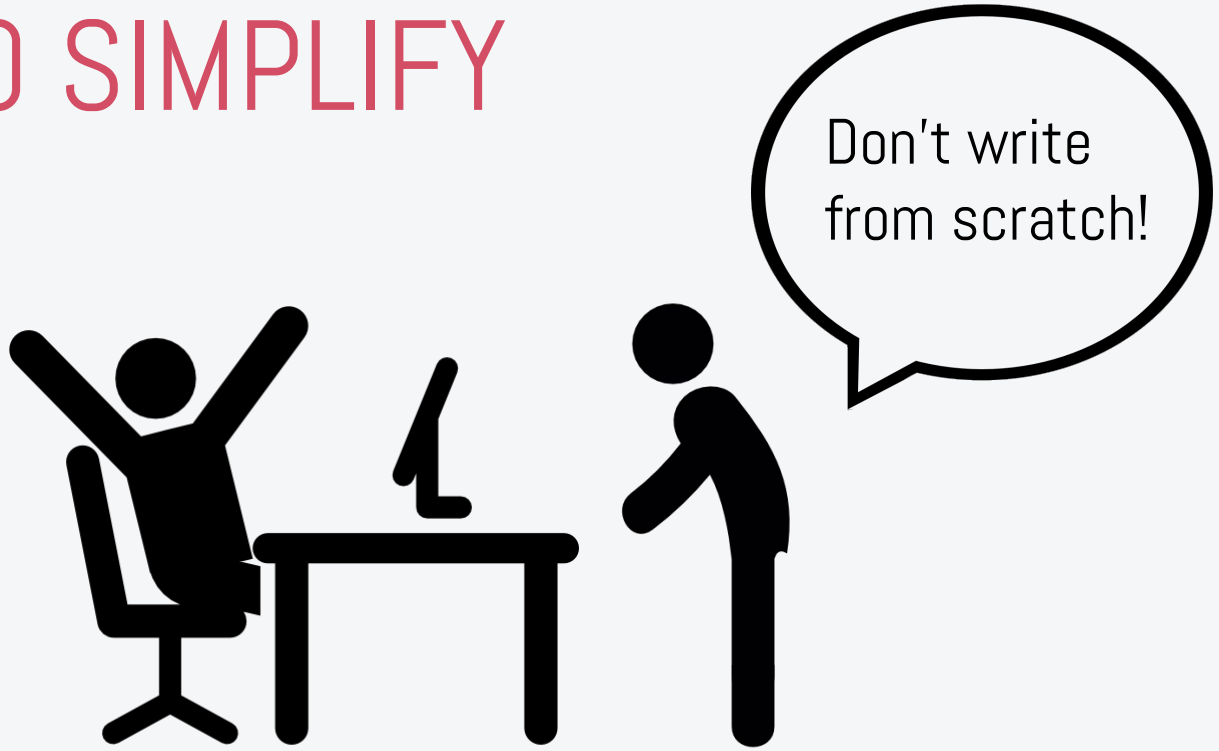


Input → Tokens → Tree → Output

TO DO LIST

1. So
2. Many
3. Things

HOW TO SIMPLIFY



USE SOME

TOOLS AND LIBRARIES

```
/// PropertyNameAndValueList :
///     PropertyAssignment
///     PropertyNameAndValueList , PropertyAssignment
propertyNameAndValueList
 : propertyAssignment ( ',' propertyAssignment )*
 ;

/// PropertyAssignment :
///     PropertyName : AssignmentExpression
///     get PropertyName ( ) { FunctionBody }
///     set PropertyName ( PropertySetParameterList ) { FunctionBody }
propertyAssignment
 : propertyName ':' singleExpression                          # PropertyExpressionAssignment
 | getter '(' ')' '{' functionBody '}'                        # PropertyGetter
 | setter '(' propertySetParameterList ')' '{' functionBody '}' # PropertySetter
 ;

/// PropertyName :
///     IdentifierName
///     StringLiteral
///     NumericLiteral
propertyName
 : identifierName
 | StringLiteral
 | numericLiteral
 ;
```

```
412   /// PropertyNameAndValueList :
413   ///     PropertyAssignment
414   ///     PropertyNameAndValueList , PropertyAssignment
415   propertyNameAndValueList
416    : propertyAssignment ( ',' propertyAssignment )*
417    ;
418
419   /// PropertyAssignment :
420   ///     PropertyName : AssignmentExpression
421   ///     get PropertyName ( ) { FunctionBody }
422   ///     set PropertyName ( PropertySetParameterList ) { FunctionBody }
423   propertyAssig
424    : propertyNa                                          # PropertyExpressionAssignment
425    | getter '('                                          # PropertyGetter
426    | setter '('                    unctionBody '}'       # PropertySetter
427    ;
428
429   /// PropertyN
430   ///     Ident
431   ///     Strin
432   ///     NumericLite
433   propertyName
434    : identifierName
435    | StringLiteral
436    | numericLiteral
437    ;
```

propertyName
 : identifierName

```
412   /// PropertyNameAndValueList :
413   ///     PropertyAssignment
414   ///     PropertyNameAndValue
415   propertyNameAndValueList
416    : propertyAssignment ( ',' 
417    ;
418
419   /// PropertyAssignment :
420   ///     PropertyName : Assig
421   ///     get PropertyName ( )
422   ///     set PropertyName ( P
423   propertyAssignment
424    : propertyName ':' singleExpression                          # PropertyExpressionAssignment
425    | getter '(' ')' '{' functionBody '}'                        # PropertyGetter
426    | setter '(' propertySetParameterList ')' '{' functionBody '}' # PropertySetter
427    ;
428
429   /// PropertyName :
430   ///     IdentifierName
431   ///     StringLiteral
432   ///     NumericLiteral
433   propertyName
434    : identifierName
435    | StringLiteral
436    | numericLiteral
437    ;
```

```
propertyAssignment
 : propertyName ':' singleExpression
```

NOT SIMPLE ENOUGH...


node
JS
+

ANTLR

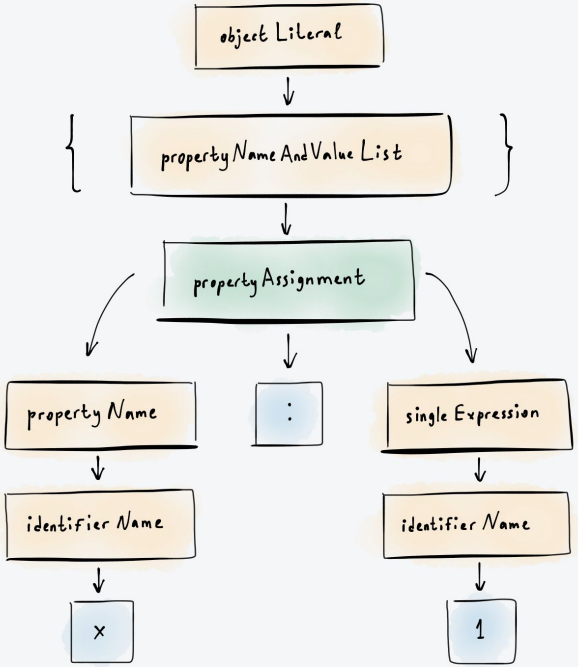
- Another Tool for Language Recognition
- Generates parser basing on grammars
- Written in Java, but has JavaScript runtime

PARSE TREE VS. AST

* AST - Abstract Syntax Tree

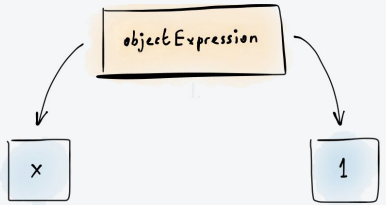
Parse Tree

- Represents concrete syntax
- Contain unusable information



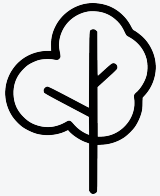
Abstract Syntax Tree

- Represents abstract syntax
- Contain only meaningful information

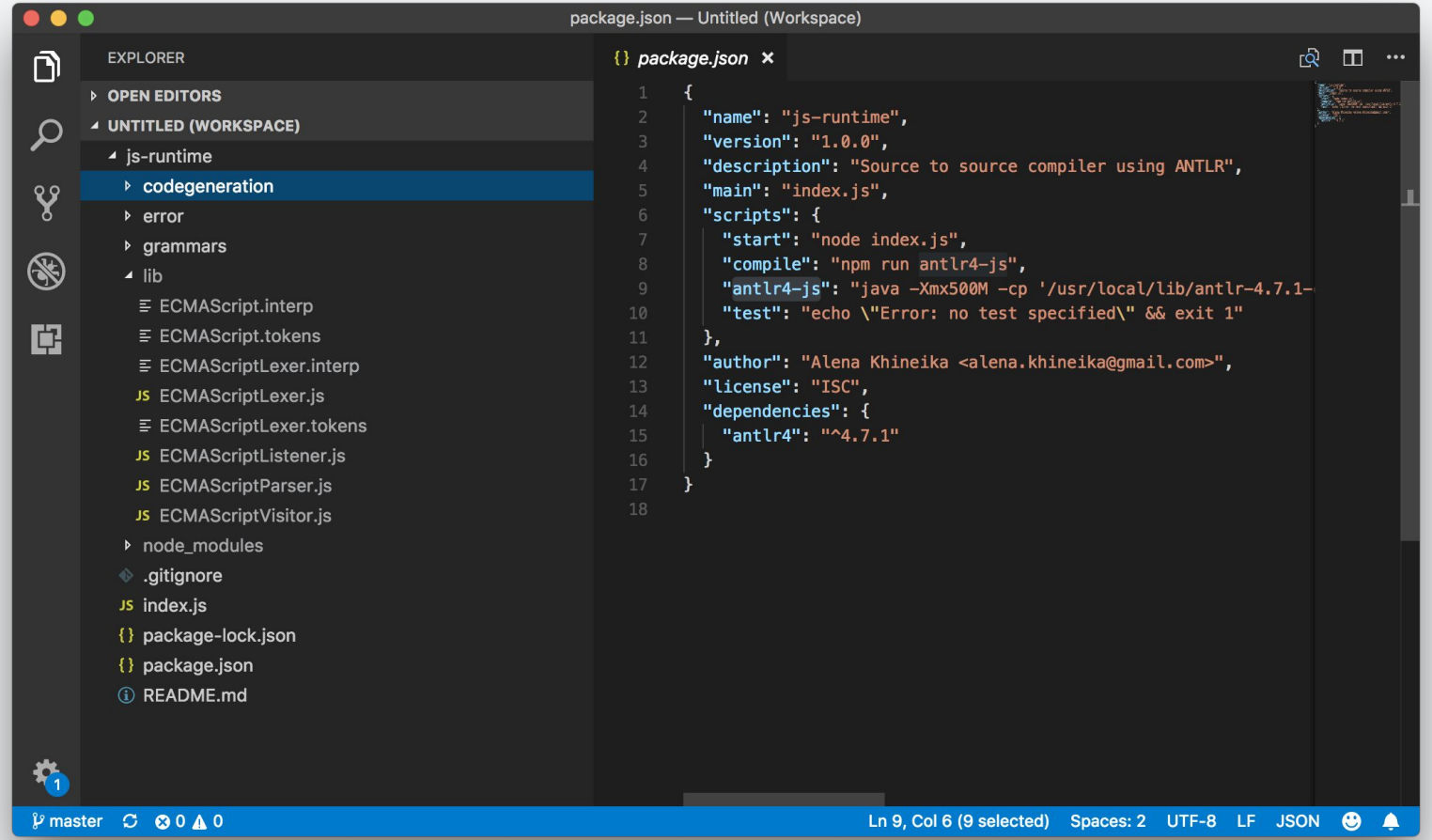


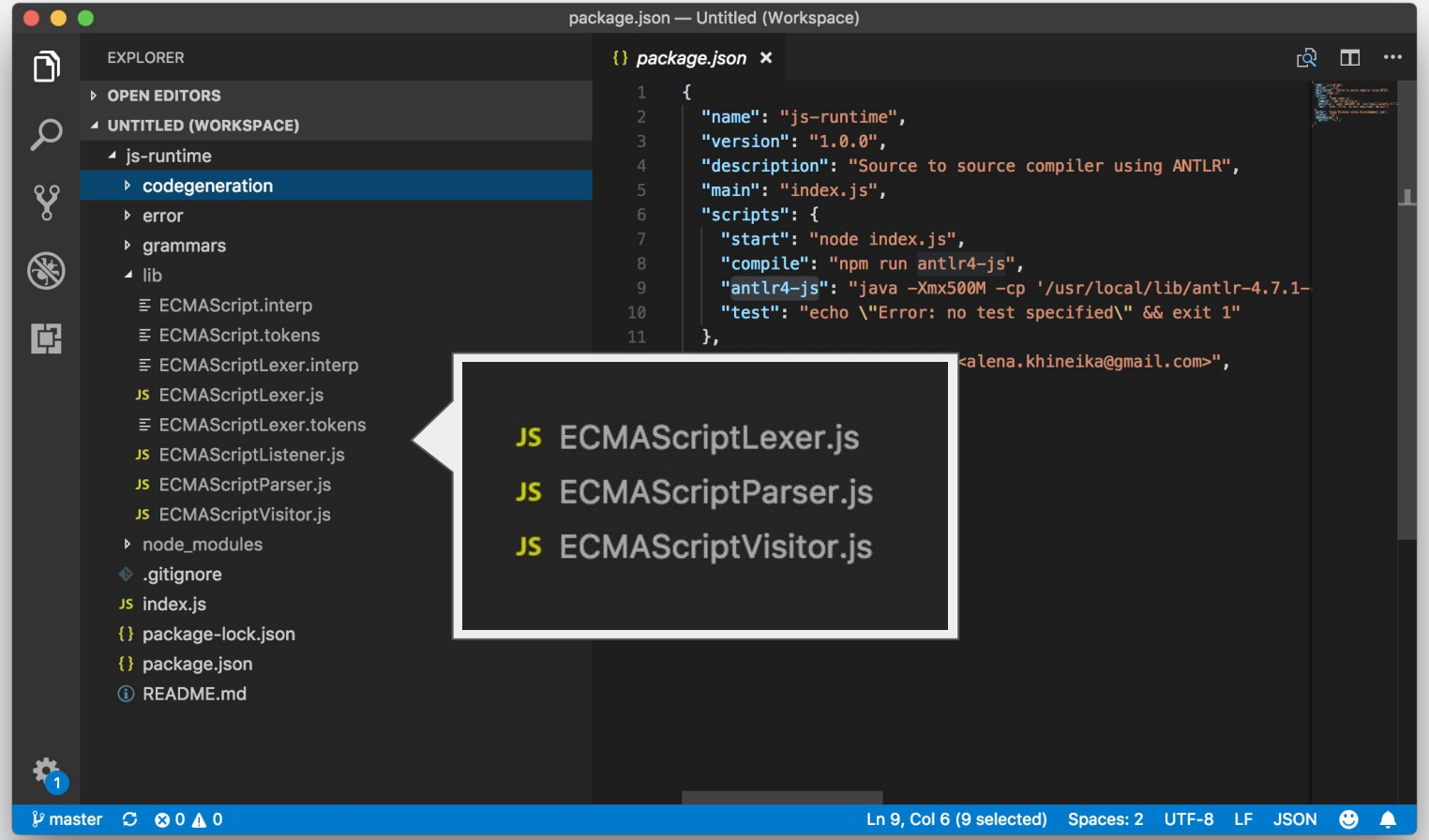
BREAKPOINT

JAVASCRIPT INPUT + ANTLR 4 = 

 + 

= PYTHON OUTPUT


package.json — Untitled (Workspace)
EXPLORER
OPEN EDITORS
UNTITLED (WORKSPACE)
js-runtime
codegeneration
error
grammars
lib
ECMAScript.interp
ECMAScript.tokens
ECMAScriptLexer.interp
ECMAScriptLexer.js
ECMAScriptLexer.tokens
ECMAScriptListener.js
ECMAScriptParser.js
ECMAScriptVisitor.js
node_modules
.gitignore
index.js
package-lock.json
package.json
README.md
package.json
{
"name": "js-runtime",
"version": "1.0.0",
"description": "Source to source compiler using ANTLR",
"main": "index.js",
"scripts": {
"start": "node index.js",
"compile": "npm run antlr4-js",
"antlr4-js": "java -Xmx500M -cp '/usr/local/lib/antlr-4.7.1-
"test": "echo \"Error: no test specified\" && exit 1"
},
"author": "Alena Khineika <alena.khineika@gmail.com>",
"license": "ISC",
"dependencies": {
"antlr4": "^4.7.1"
}
}
master
0 0
Ln 9, Col 6 (9 selected)
Spaces: 2
UTF-8
LF
JSON

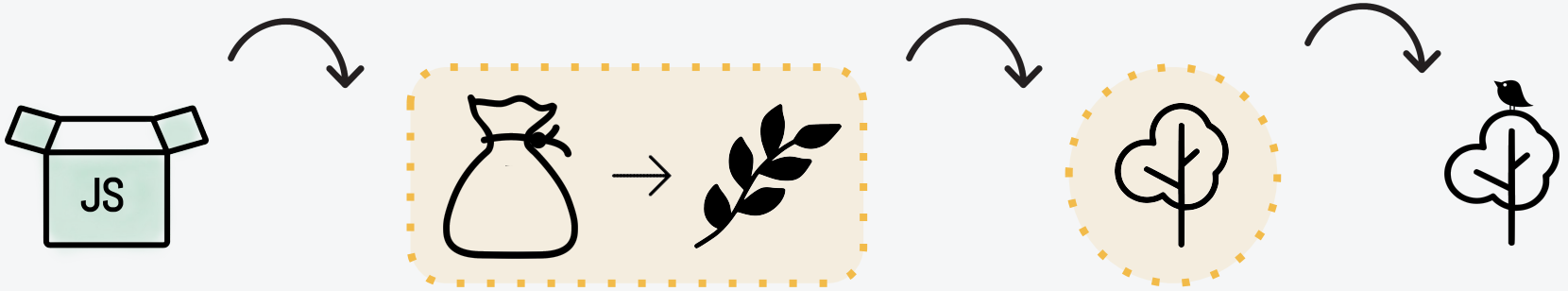

package.json — Untitled (Workspace)
EXPLORER
OPEN EDITORS
UNTITLED (WORKSPACE)
js-runtime
codegeneration
error
grammars
lib
ECMAScript.interp
ECMAScript.tokens
ECMAScriptLexer.interp
ECMAScriptLexer.js
ECMAScriptLexer.tokens
ECMAScriptListener.js
ECMAScriptParser.js
ECMAScriptVisitor.js
node_modules
.gitignore
index.js
package-lock.json
package.json
README.md
package.json
{
"name": "js-runtime",
"version": "1.0.0",
"description": "Source to source compiler using ANTLR",
"main": "index.js",
"scripts": {
"start": "node index.js",
"compile": "npm run antlr4-js",
"antlr4-js": "java -Xmx500M -cp '/usr/local/lib/antlr-4.7.1-
"test": "echo \"Error: no test specified\" && exit 1"
},
<alena.khineika@gmail.com>",
ECMAScriptLexer.js
ECMAScriptParser.js
ECMAScriptVisitor.js
master
0 0
Ln 9, Col 6 (9 selected)
Spaces: 2
UTF-8
LF
JSON

ANTLR AUXILIARY FILES

*Lexer.js
Lexical Analysis

*Parser.js
Syntax Analysis

*Visitor.js
Tree Traversal



Input

Characters

Tokens

Tree

New Tree

JS index.js ✕

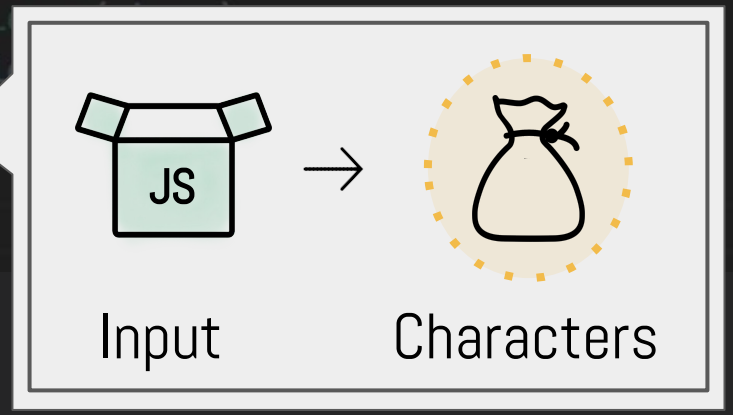
```
1   const antlr4 = require('antlr4');
2   const ECMAScriptLexer = require('./lib/ECMAScriptLexer.js');
3   const ECMAScriptParser = require('./lib/ECMAScriptParser.js');
4
5   const input = '{x: 1}';
6
7   const chars = new antlr4.InputStream(input);
8   const lexer = new ECMAScriptLexer.ECMAScriptLexer(chars);
9   const tokens = new antlr4.CommonTokenStream(lexer);
10  const parser = new ECMAScriptParser.ECMAScriptParser(tokens);
11
12  const tree = parser.expressionSequence();
```

JS index.js ✕

```
1   const antlr4 = require('antlr4');
2   const ECMAScriptLexer = require('./lib/ECMAScriptLexer.js');
3   const ECMAScriptParser = require('./lib/ECMAScriptParser.js');
4
5   const input = '{x: 1}';
6
7   const chars = new antlr4.InputStream(input);
8   const lexer = new ECMAScriptLexer.ECMAScriptLexer(chars);
9   const tokens = new antlr4.CommonTokenStream(lexer);
10  const parser = new ECMAScriptParser.ECMAScriptParser(tokens);
11
12  const tree = parser.expressionSequence();
```

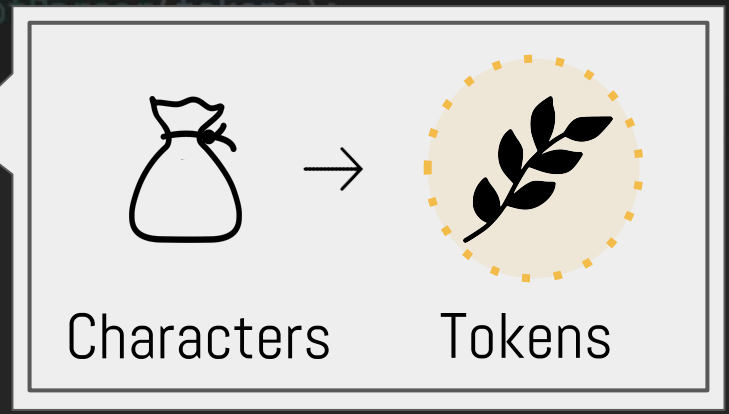
JS index.js ×

```
1   const antlr4 = require('antlr4');
2   const ECMAScriptLexer = require('./lib/ECMAScriptLexer.js');
3   const ECMAScriptParser = require('./lib/ECMAScriptParser.js');
4
5   const input = '{x: 1}';
6
7   const chars = new antlr4.InputStream(input);
8   const lexer = new ECMAScriptLexer.ECMAScriptL
9   const tokens = new antlr4.CommonTokenStream(l
10  const parser = new ECMAScriptParser.ECMAScr
11
12  const tree = parser.expressionSequence();
```



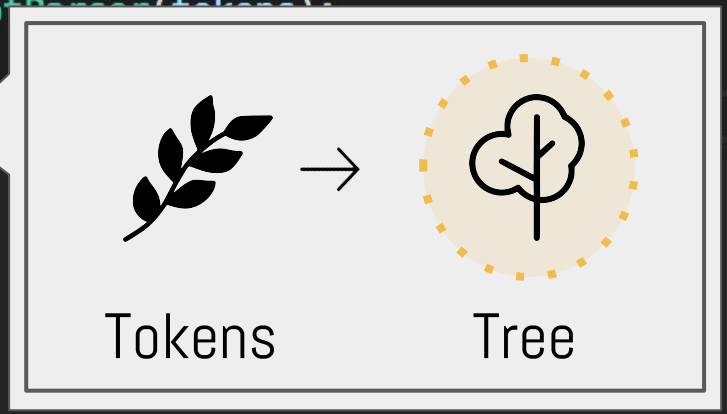
JS index.js ×

```
1   const antlr4 = require('antlr4');
2   const ECMAScriptLexer = require('./lib/ECMAScriptLexer.js');
3   const ECMAScriptParser = require('./lib/ECMAScriptParser.js');
4
5   const input = '{x: 1}';
6
7   const chars = new antlr4.InputStream(input);
8   const lexer = new ECMAScriptLexer.ECMAScriptLexer(chars);
9   const tokens = new antlr4.CommonTokenStream(lexer);
10  const parser = new ECMAScriptParser.ECMAScript
11
12  const tree = parser.expressionSequence();
```



JS index.js

```
1   const antlr4 = require('antlr4');
2   const ECMAScriptLexer = require('./lib/ECMAScriptLexer.js');
3   const ECMAScriptParser = require('./lib/ECMAScriptParser.js');
4
5   const input = '{x: 1}';
6
7   const chars = new antlr4.InputStream(input);
8   const lexer = new ECMAScriptLexer.ECMAScriptLexer(chars);
9   const tokens = new antlr4.CommonTokenStream(lexer);
10  const parser = new ECMAScriptParser.ECMAScript
11
12  const tree = parser.expressionSequence();
```



Alena Khineika in DailyJS
Jul 3 · 15 min

Compiler in JavaScript using ANTLR

> *This article can be viewed as both a practical guide for writing a JavaScript compiler and a theoretical resource that describes the basic concepts and principles of compiler design.*

Early this year, I joined the team which develops and supports MongoDB Compass, a graphical…

http://bit.ly/compiler-in-javascript

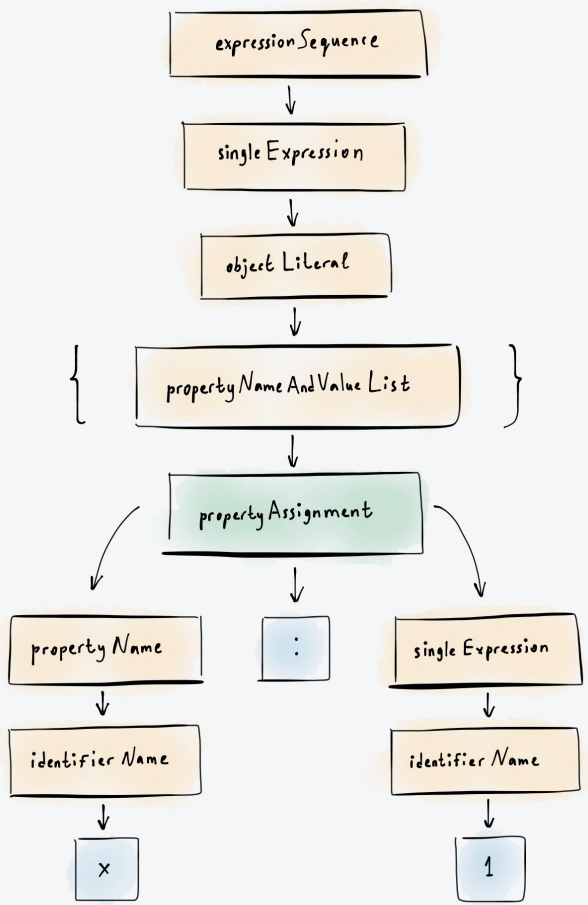
ECMAScript.interp

```
rule names:
program
sourceElements
sourceElement
statement
block
statementList
variableStatement
variableDeclarationList
variableDeclaration
initialiser
emptyStatement
expressionStatement
ifStatement
```

index.js

```
const chars = new antlr4.InputStream(input);
const lexer = new ECMAScriptLexer.ECMAScriptLexer(chars);
const tokens = new antlr4.CommonTokenStream(lexer);
const parser = new ECMAScriptParser.ECMAScriptParser(tokens);

const tree = parser.expressionSequence();
```



VISITOR VS. LISTENER

Visitor

- You should explicitly call child methods
- Methods can return any custom type
- You can modify existing nodes

```
↓ visitPropertyNameAndValue(ctx)
↓ visitPropertyAssignment(ctx)
↓ visitPropertyName(ctx)
↓ visitIdentifierName(ctx)
↓ visitTerminal(ctx)
```

Listener

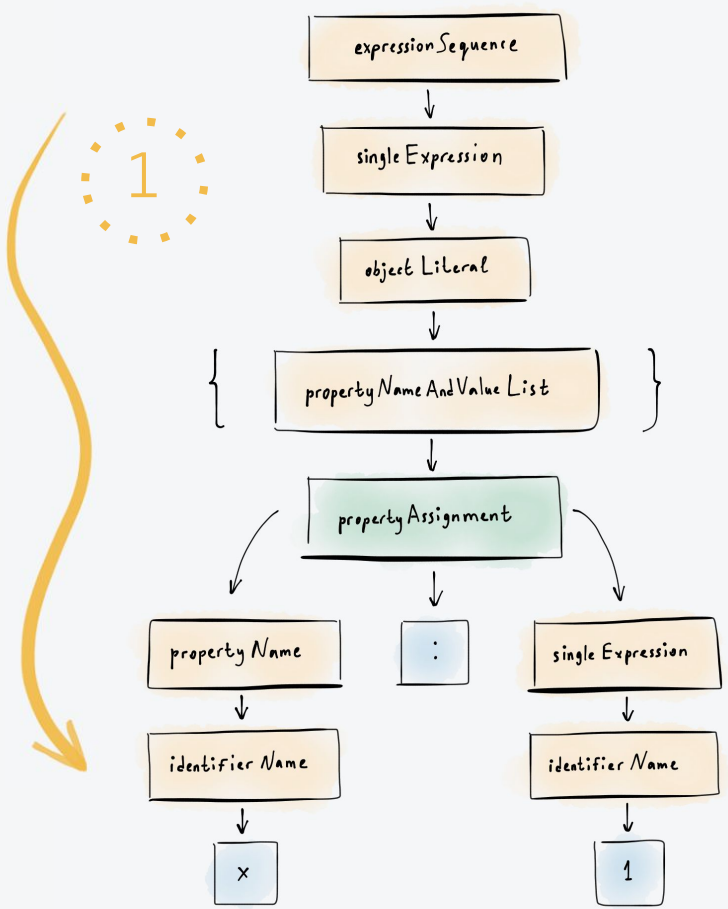
- Methods are called automatically
- Methods can't return a value
- You should store values outside the tree

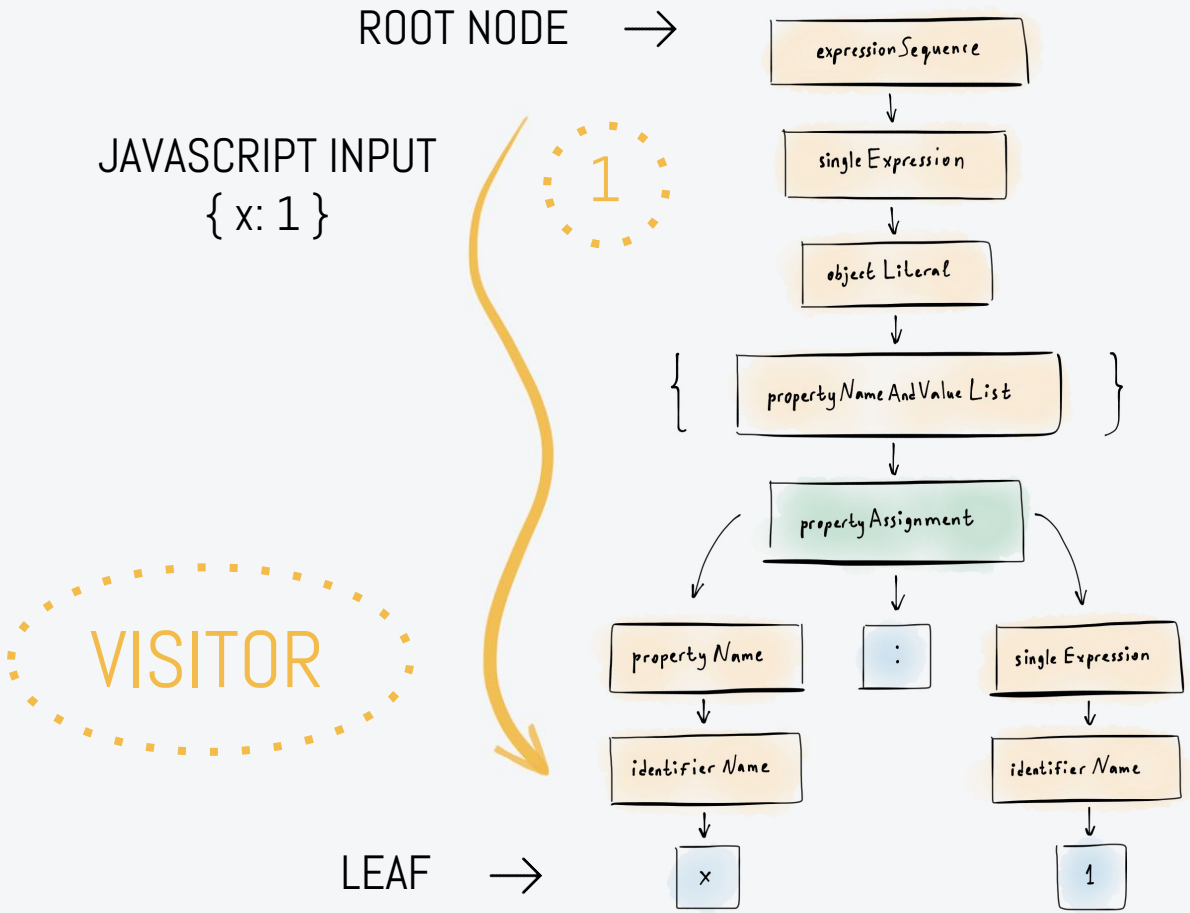
```
→ enterPropertyAssignment(ctx)
  → enterPropertyName(ctx)
    → enterIdentifierName(ctx)
      → enterTerminal(ctx)
      ← exitTerminal(ctx)
    ← exitIdentifierName(ctx)
  ← exitPropertyName(ctx)
← exitPropertyAssignment(ctx)
```

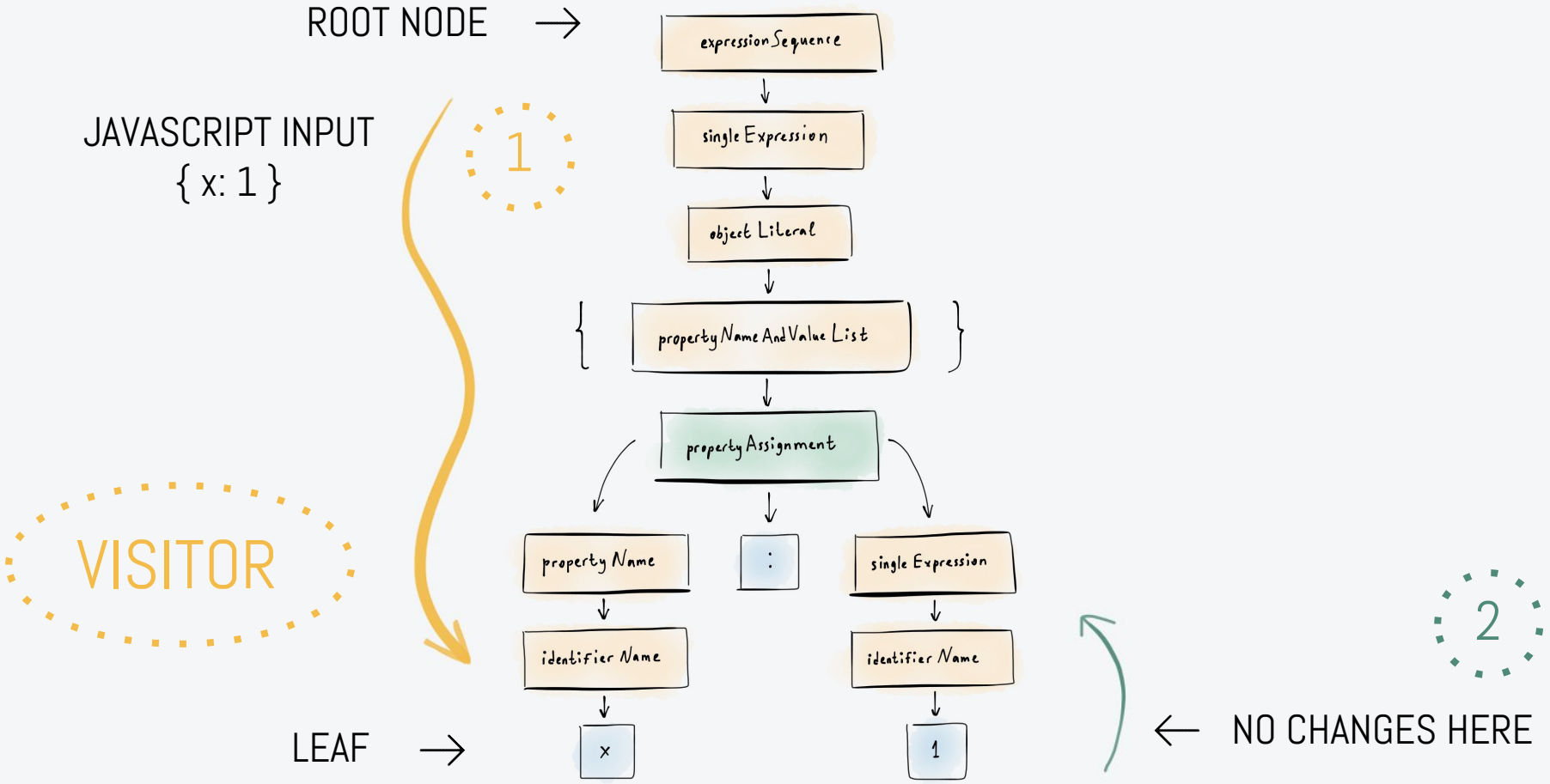
JAVASCRIPT INPUT

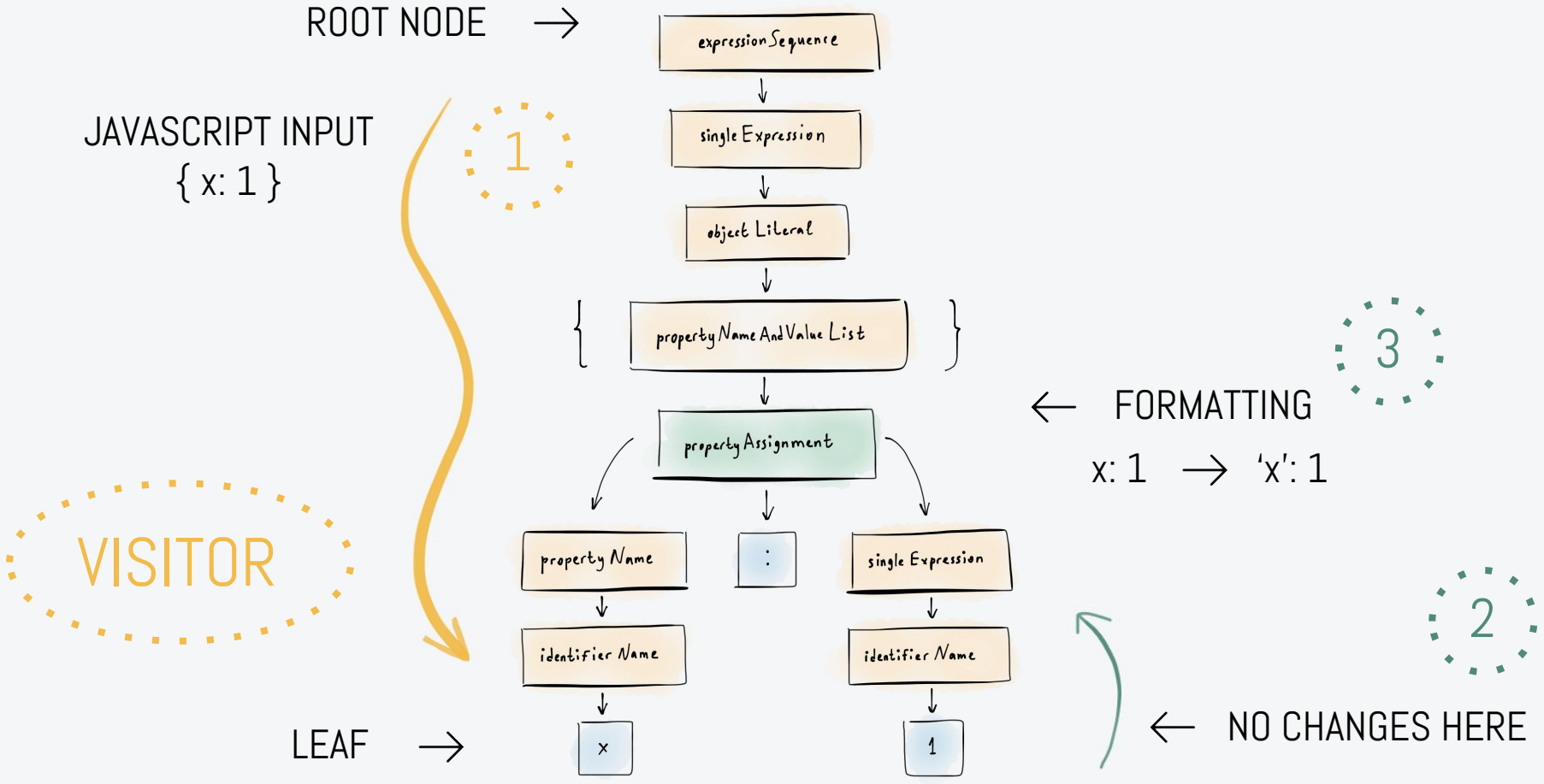
{ x: 1 }

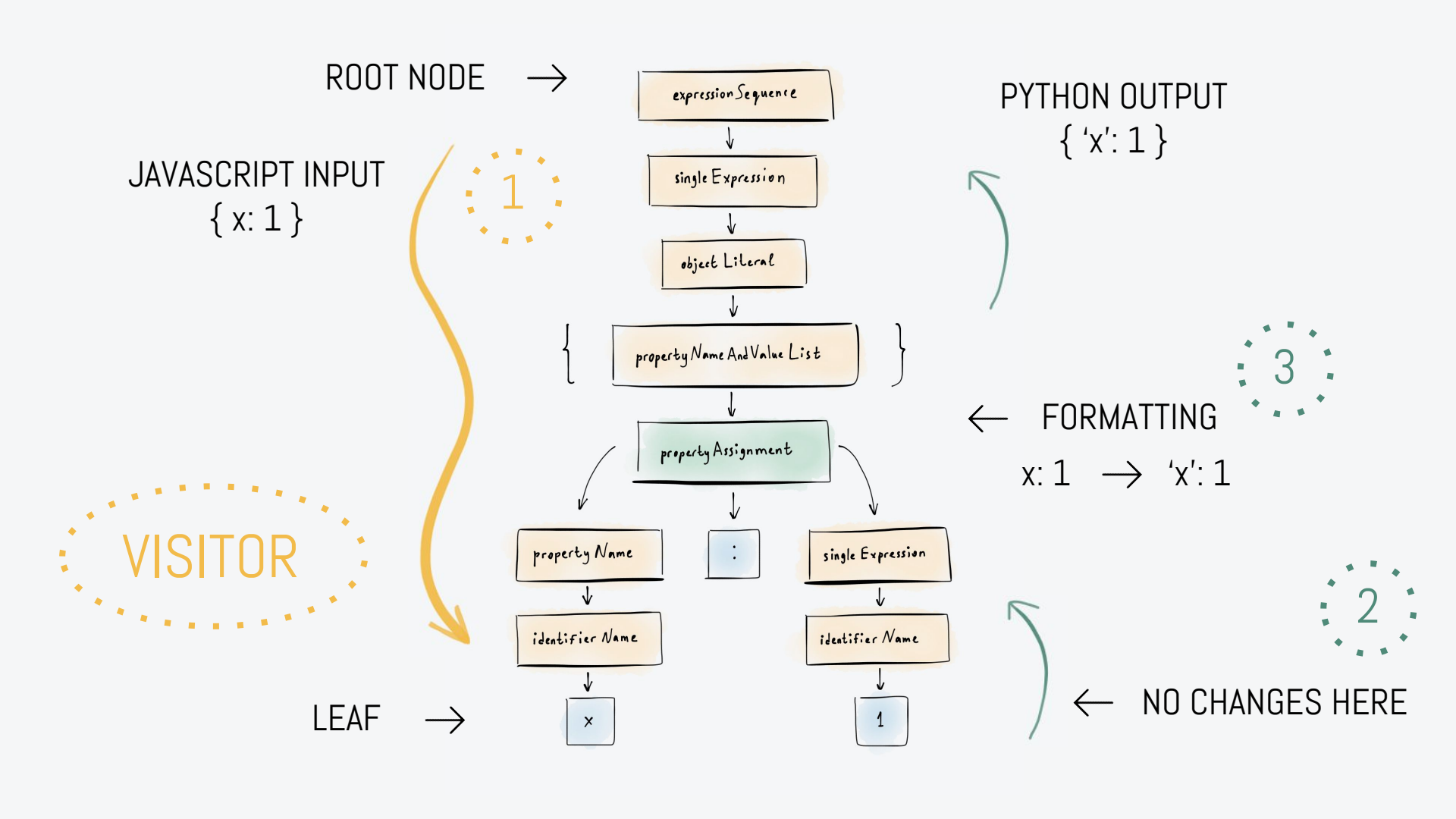
VISITOR




ROOT NODE
JAVASCRIPT INPUT
{ x: 1 }
1
VISITOR
LEAF
expressionSequence
singleExpression
objectLiteral
{
propertyNameAndValueList
}
propertyAssignment
propertyName
:
singleExpression
identifierName
identifierName
x
1


ROOT NODE
JAVASCRIPT INPUT
{ x: 1 }
1
expressionSequence
singleExpression
objectLiteral
{
propertyNameAndValueList
}
propertyAssignment
propertyName
:
singleExpression
identifierName
identifierName
x
1
VISITOR
LEAF
2
NO CHANGES HERE


ROOT NODE
expressionSequence
JAVASCRIPT INPUT
{ x: 1 }
1
singleExpression
objectLiteral
{
propertyNameAndValueList
}
3
FORMATTING
propertyAssignment
x: 1
'x': 1
VISITOR
propertyName
:
singleExpression
identifierName
identifierName
2
LEAF
x
1
NO CHANGES HERE


ROOT NODE
expressionSequence
singleExpression
objectLiteral
{
propertyNameAndValueList
}
propertyAssignment
propertyName
:
singleExpression
identifierName
identifierName
x
1
LEAF
JAVASCRIPT INPUT
{ x: 1 }
1
PYTHON OUTPUT
{ 'x': 1 }
3
FORMATTING
x: 1 → 'x': 1
2
NO CHANGES HERE
VISITOR

JS ECMAScriptVisitor.js

```
var antlr4 = require('antlr4/index');

function ECMAScriptVisitor() {
  antlr4.tree.ParseTreeVisitor.call(this);
  return this;
}

ECMAScriptVisitor.prototype = Object.create(antlr4.tree.ParseTreeVisitor.prototype);
ECMAScriptVisitor.prototype.constructor = ECMAScriptVisitor;

ECMAScriptVisitor.prototype.visitProgram = function(ctx) {
  return this.visitChildren(ctx);
};

ECMAScriptVisitor.prototype.visitPropertyExpressionAssignment = function(ctx) {
  return this.visitChildren(ctx);
};
```

Not here!

JS PythonGenerator.js

```
const ECMAScriptVisitor = require('../lib/ECMAScriptVisitor').ECMAScriptVisitor;

class Visitor extends ECMAScriptVisitor {}

module.exports = Visitor;
```

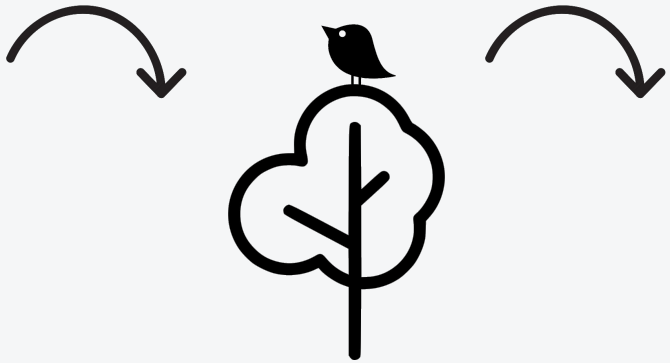
Here!

JS PythonGenerator.js ●

```
1  const ECMAScriptVisitor = require('../lib/ECMAScriptVisitor')
2    .ECMAScriptVisitor;
3
4  class Visitor extends ECMAScriptVisitor {
5    start(ctx) {
6      return this.visitExpressionSequence(ctx);
7    }
8
9    visitPropertyExpressionAssignment(ctx) {
10     const key = this.visit(ctx.propertyName());
11     const value = this.visit(ctx.singleExpression());
12
13     return `'${key}': ${value}`;
14   }
15
16   visitTerminal(ctx) {
17     return ctx.getText();
18   }
19 }
20
21 module.exports = Visitor;
22
```

TRANSFORMATION

JAVASCRIPT
{ x: 1 }



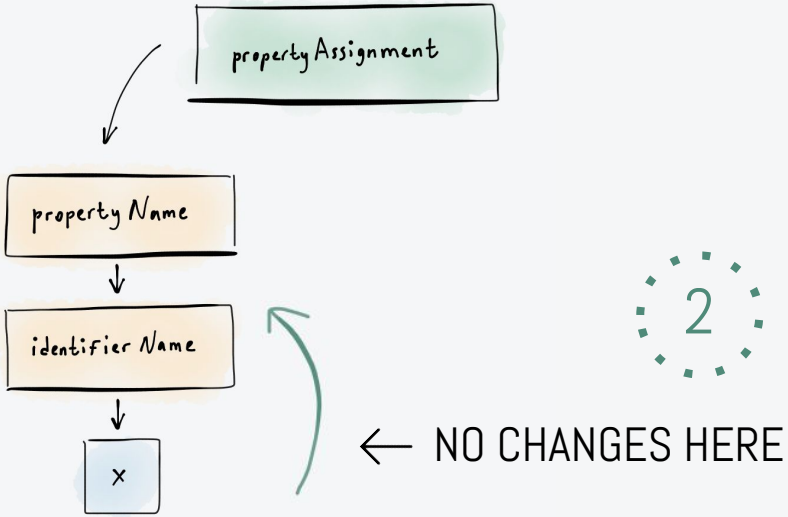
PYTHON
{ 'x': 1 }

JS PythonGenerator.js ●

```
1  const ECMAScriptVisitor = require('../lib/ECMAScriptVisitor')
2    .ECMAScriptVisitor;
3
4  class Visitor extends ECMAScriptVisitor {
5    start(ctx) {
6      return this.visitExpressionSequence(ctx);
7    }
8
9    visitPropertyExpressionAssignment(ctx) {
10     const key = this.visit(ctx.propertyName());
11     const value = this.visit(ctx.singleExpression());
12
13     return `'${key}': ${value}`;
14   }
15
16   visitTerminal(ctx) {
17     return ctx.getText();
18   }
19 }
20
21 module.exports = Visitor;
22
```

Visiting the left and the right side of the tree separately

LEFT HAND SIDE VISITING



* The same logic is applicable for the right side

JS PythonGenerator.js

```
3  class Visitor extends ECMAScriptVisitor {
4    start(ctx) {
5      return this.visitExpressionSequence(ctx);
6    }
7
8    visitPropertyExpressionAssignment(ctx) {
9      const key = this.visit(ctx.propertyName());
10     const value = this.visit(ctx.singleExpression());
11
12     return `'${key}': ${value}`;
13   }
14
15   visitTerminal(ctx) {
16     return ctx.getText();
17   }
18 }
19
20 module.exports = Visit
21
```

Standard methods:
visit()
visitTerminal()

JS PythonGenerator.js ●

```
1  const ECMAScriptVisitor = require('../lib/ECMAScriptVisitor')
2    .ECMAScriptVisitor;
3
4  class Visitor extends ECMAScriptVisitor {
5    start(ctx) {
6      return this.visitExpressionSequence(ctx);
7    }
8
9    visitPropertyExpressionAssignment(ctx) {
10     const key = this.visit(ctx.propertyName());
11     const value = this.visit(ctx.singleExpression());
12
13     return `'${key}': ${value}`;
14   }
15
16   visitTerminal(ctx) {
17     return ctx.getText();
18   }
19 }
20
21 module.exports = Visitor;
22
```

Formatting 3

x: 1 → 'x': 1

SINGLE AND DOUBLE QUOTES

What do we support?

DIFFERENT NUMERAL SYSTEMS

Answer should be accurate

USER INPUT FORMAT

Spaces, empty lines etc.

ESCAPE SEQUENCES

Have to be escaped

FLOATING POINT NUMBERS

Don't damage Pi

COMMENTS

Yes / No

```
608     | singleExpression '=' expressionSequence                 # AssignmentExpression
609     | singleExpression assignmentOperator expressionSequence  # AssignmentOperatorExpression
610     | This                                                    # ThisExpression
611     | Identifier                                              # IdentifierExpression
612     | Number arguments                                        # NumberExpression
613     | literal                                                 # LiteralExpression
614     | arrayLiteral                                            # ArrayLiteralExpression
615     | objectLiteral                                           # ObjectLiteralExpression
616     | '(' expressionSequence ')'                              # ParenthesizedExpression
617     ;
...
835 Function   : 'function';
836 This       : 'this';
837 With       : 'with';
838 Default    : 'default';
839 If         : 'if';
840 Throw      : 'throw';
841 Delete     : 'delete';
842 In         : 'in';
843 Try        : 'try';
844 Number     : 'Number';
845
846 /// 7.6.1.2 Future Reserved Words
847 Class   : 'class';
```

JS PythonGenerator.js ●

```
8
9     visitPropertyExpressionAssignment(ctx) {
10      const key = this.visit(ctx.propertyName());
11      const value = this.visit(ctx.singleExpression());
12
13      return `'${key}': ${value}`;
14    }
15
16    visitTerminal(ctx) {
17      return ctx.getText();
18    }
19
20    visitNumberExpression(ctx) {
21      const argumentList = ctx.arguments().argumentList();
22      const arg = argumentList.singleExpression()[0];
23      const number = this.visit(arg);
24
25      return `int(${number})`;
26    }
27  }
28
29  module.exports = Visitor;
30
```

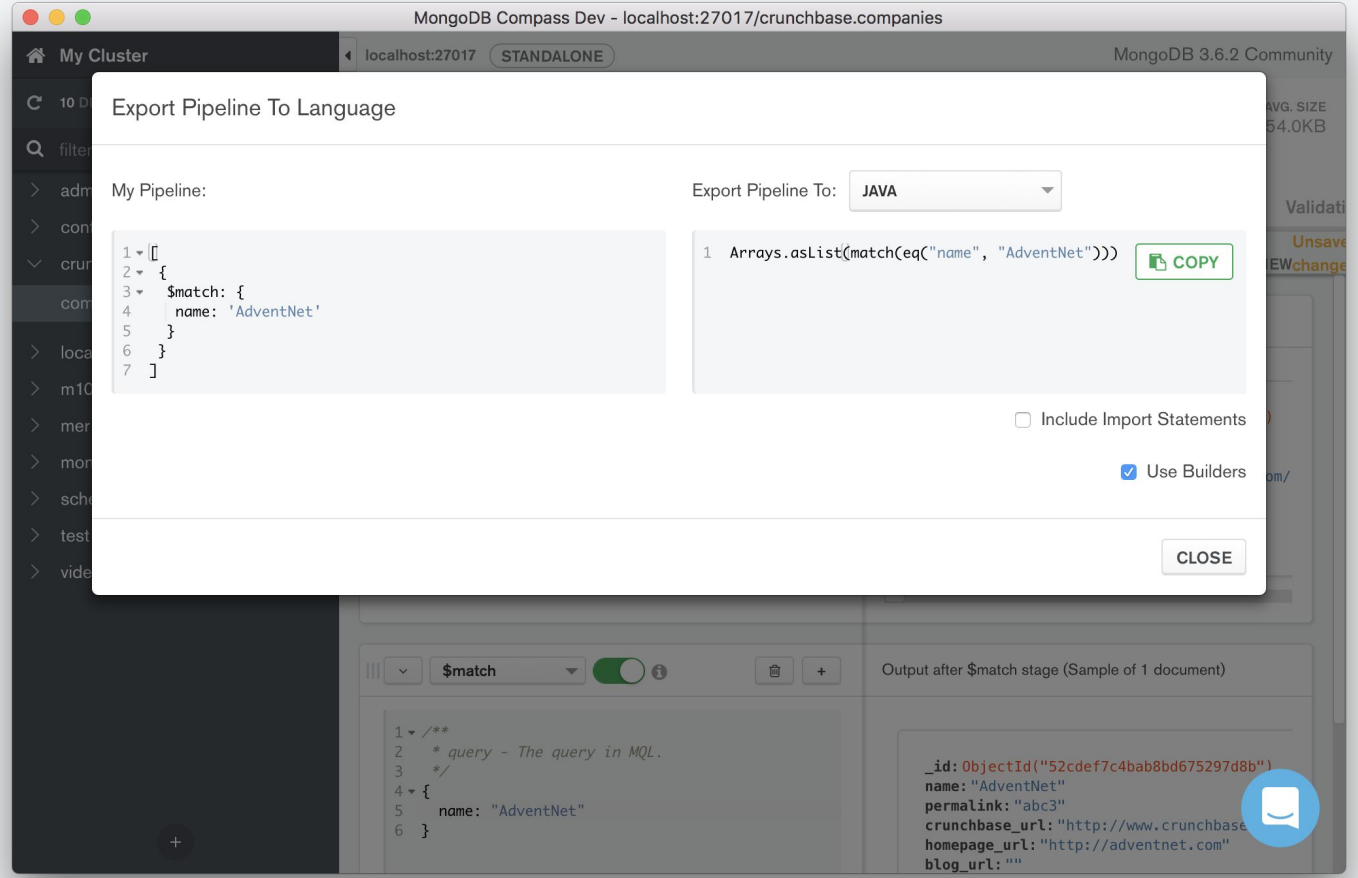
JavaScript input:

{ x: Number(1) }

Python output:

{ 'x': int(1) }

COMPILERS ARE AWESOME!


MongoDB Compass Dev - localhost:27017/crunchbase.companies
My Cluster
localhost:27017
STANDALONE
MongoDB 3.6.2 Community
Export Pipeline To Language
My Pipeline:
[
{
$match: {
name: 'AdventNet'
}
}
]
Export Pipeline To:
JAVA
Arrays.asList(match(eq("name", "AdventNet")))
COPY
Include Import Statements
Use Builders
CLOSE
$match
Output after $match stage (Sample of 1 document)
/**
* query - The query in MQL.
*/
{
name: "AdventNet"
}
_id: ObjectId("52cdef7c4bab8bd675297d8b")
name: "AdventNet"
permalink: "abc3"
homepage_url: "http://adventnet.com"
blog_url: ""

ALENA KHINEIKA

@alena_khineika

@alena_khineika

@alenakhineika