

# //TODO

Write linting rules to enforce  
your team's code conventions



ESLint

Write your own javascript  
transpiling code



Write powerful “code-mods”  
to automatically refactor  
thousands of legacy scripts  
from ES5 to ES6


jscodeshift

ESTree-  
based AST

+

Visitor  
Pattern

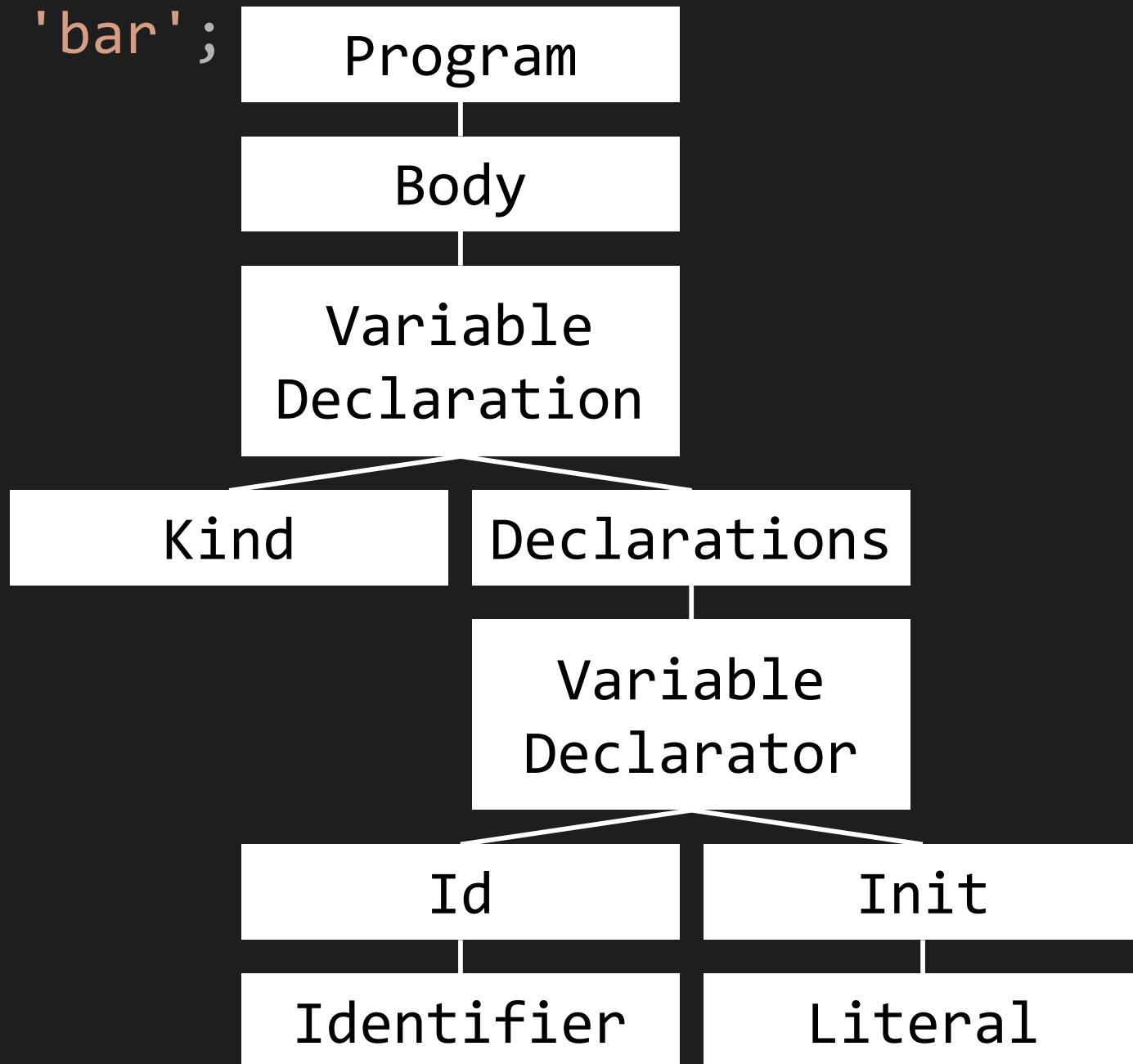




# Abstract Syntax Tree



```
var foo = 'bar';
```



```
var foo = 'bar'; {
  "type": "Program",
  "body": [
    {
      "type": "VariableDeclaration",
      "declarations": [
        {
          "type": "VariableDeclarator",
          "id": {
            "type": "Identifier",
            "name": "foo"
          },
          "init": {
            "type": "Literal",
            "value": "bar"
          }
        }
      ],
      "kind": "var"
    }
  ]
}
```

```
var foo = 'bar'; {
  "type": "Program",
  "body": [
    {
      "type": "VariableDeclaration",
      "declarations": [
        {
          "type": "VariableDeclarator",
          "id": {
            "type": "Identifier",
            "name": "foo"
          },
          "init": {
            "type": "Literal",
            "value": "bar"
          }
        }
      ],
      "kind": "var"
    }
  ]
}
```

**I/** { Parse  
Traverse

**O** { Manipulate  
Generate code

**I/ { Parse**

**O {**

AST Explorer

Save

Fork

JavaScript

</> acorn

Parser Settings

Transform

Help

```
1 /**
2  * Paste or drop some JavaScript here and explore
3  * the syntax tree created by chosen parser.
4  * You can use all the cool new features from ES6
5  * and even more. Enjoy!
6  */
7
8 let tips = [
9   "Click on any AST node with a '+' to expand it",
10
11   "Hovering over a node highlights the \
12    corresponding part in the source code",
13
14   "Shift click on an AST node expands the whole subtree"
15 ];
16
17 function printTips() {
18   tips.forEach((tip, i) => console.log(`Tip ${i}:` + tip));
19 }
20
```

TreeJSONParser: acorn

☒ Autofocus☒ Hide methods☐ Hide empty keys☐ Hide local data

```
- Program {
  type: "Program"
  start: 0
  end: 476
  + range: [2 elements]
  - body: [
    + VariableDeclaration {type, start, end, range,
      declarations, ... +1}
    + FunctionDeclaration {type, start, end, range,
      ... +4}
  ]
  sourceType: "module"
}
```

astexplorer.net



**W/ { Traverse**

**O {**



# ESLint rule

This code is a simplified version of the `eslint-plugin-piggyback` plugin:  
<https://github.com/cowchimp/eslint-plugin-piggyback>

```
1. module.exports = function(context) {
2.   var globalScope;
3.
4.   return {
5.     "MemberExpression": function(node) {
6.       if(
7.         node.object.type === "Identifier" &&
8.         node.object.name === "window" &&
9.         !isValidGlobalProperty(node.property.name)
10.      ) {
```

```
1. module.exports = function(context) {
2.   var globalScope;
3.
4.   return {
5.     "MemberExpression": function(node) {
6.       if(
7.         node.object.type === "Identifier" &&
8.         node.object.name === "window" &&
9.         !isValidGlobalProperty(node.property.name)
10.      ) {
11.        context.report(
```

Visit all `{{foo}}.{{bar}}` nodes

```
14.         `${node.object.name} global scope`,
```

```
1. module.exports = function(context) {
2.   var globalScope;
3.
4.   return {
5.     "MemberExpression": function(node) {
6.       if(
7.         node.object.type === "Identifier" &&
8.         node.object.name === "window" &&
9.         !isValidGlobalProperty(node.property.name)
10.      ) {
11.        context.report(
12.          node,
13.          `${node.property.name} piggybacks on the
14.          ${node.object.name} global scope`,
15.          ).
```

Check if it's a naughty node



```
3.
4.     return {
5.         "MemberExpression": function(node) {
6.             if(
7.                 node.object.type === "Identifier" &&
8.                 node.object.name === "window" &&
9.                 !isValidGlobalProperty(node.property.name)
10.            ) {
11.                context.report(
12.                    node,
13.                    `${node.property.name} piggybacks on the
14.                    ${node.object.name} global scope`,
15.                );
16.            }
17.        },
18.
19.        "Program": function() {
```

**Report the violation**

```
22.     };
```

```
14.         `${node.object.name} global scope`,
15.     );
16. }
17. },
18.
19.     "Program": function() {
20.         globalScope = context.getScope();
21.     }
22. };
23.
24. function isValidGlobalProperty(propertyName) {
25.     return globalScope.variables.some(
26.         v => v.name == propertyName
27.     );
28. }
29. };
```

## Querying scope



VISITOR



**I/ {**

**O { Manipulate**

*BABEL*



# Babel plugin

```
1. // Babel babel-plugin-transform-remove-debugger
2. // Author: Sebastian McKenzie
3. // License: MIT
4. // Url: https://github.com/babel
5.
6. export default function () {
7.   return {
8.     visitor: {
9.       "DebuggerStatement": function(path) {
10.         path.remove();
11.       }
12.     }
13.   }
14. }
```

```
1. // Babel babel-plugin-transform-remove-debugger
2. // Author: Sebastian McKenzie
3. // License: MIT
4. // Url: https://github.com/babel
5.
6. export default function () {
7.   return {
8.     visitor: {
9.       "DebuggerStatement": function(path) {
10.         path.remove();
11.       }
12.     }
13.   };
14. }
```

Find any `debugger` statement nodes

```
1. // Babel babel-plugin-transform-remove-debugger
2. // Author: Sebastian McKenzie
3. // License: MIT
4. // Url: https://github.com/babel
5.
6. export default function () {
7.   return {
8.     visitor: {
9.       "DebuggerStatement": function(path) {
10.         path.remove();
11.       }
12.     }
13.   };
14. }
```

Remove the node from the AST

**// {**

**O {** Generate code

jscodeshift



# jscodeshift transform

```
1. module.exports = function(file, api) {  
2.   var j = api.jscodeshift;  
3.   var root = j(file.source);  
4.  
5.   root.find(j.VariableDeclaration, { kind: 'var' })  
6.     .forEach(function(p) {  
7.       var letStatement = j.variableDeclaration(  
8.         'let',  
9.         p.value.declarations  
10.       );  
11.     });  
12. }
```

```
1. module.exports = function(file, api) {  
2.   var j = api.jscodeshift;  
3.   var root = j(file.source);  
4.  
5.   root.find(j.VariableDeclaration, { kind: 'var' })  
6.     .forEach(function(p) {  
7.       var letStatement = j.variableDeclaration(  
8.         'let',  
9.         p.value.declarations  
10.      ).  
11.    }  
12.  }  
13.}
```

Convert the source code to an AST

```
1. module.exports = function(file, api) {  
2.     var j = api.jscodeshift;  
3.     var root = j(file.source);  
4.  
5.     root.find(j.VariableDeclaration, { kind: 'var' })  
6.         .forEach(function(p) {  
7.             var letStatement = j.variableDeclaration(  
8.                 'let',  
9.                 p.value.declarations  
10.            );  
11.             return j(p).replaceWith(letStatement);  
12.         })  
13. }
```

Find all `var` declaration nodes

```
15. }
```

```
1. module.exports = function(file, api) {  
2.     var j = api.jscodeshift;  
3.     var root = j(file.source);  
4.  
5.     root.find(j.VariableDeclaration, { kind: 'var' })  
6.         .forEach(function(p) {  
7.             var letStatement = j.variableDeclaration(  
8.                 'let',  
9.                 p.value.declarations  
10.            );  
11.             return j(p).replaceWith(letStatement);  
12.         });  
13.
```

...and for each one...

```
1. module.exports = function(file, api) {  
2.   var j = api.jscodeshift;  
3.   var root = j(file.source);  
4.  
5.   root.find(j.VariableDeclaration, { kind: 'var' })  
6.     .forEach(function(p) {  
7.       var letStatement = j.variableDeclaration(  
8.         'let',  
9.         p.value.declarations  
10.      );  
11.      return j(p).replaceWith(letStatement);  
12.    });  
13.  
14.   return root.toSource();  
15. };
```

Create a copy node but make it a `let` declaration



```
2.   var j = api.jscodeshift;  
3.   var root = j(file.source);  
4.  
5.   root.find(j.VariableDeclaration, { kind: 'var' })  
6.     .forEach(function(p) {  
7.       var letStatement = j.variableDeclaration(  
8.         'let',  
9.         p.value.declarations  
10.      );  
11.      return j(p).replaceWith(letStatement);  
12.    });  
13.  
14.   return root.toSource();  
15. };
```

Replace the original node with the new one

```
4.  
5.   root.find(j.VariableDeclaration, { kind: 'var'  
6.     .forEach(function(p) {  
7.       var letStatement = j.variableDeclaration(  
8.         'let',  
9.         p.value.declarations  
10.      );  
11.      return j(p).replaceWith(letStatement);  
12.    });  
13.  
14.   return root.toSource();  
15. };
```

Convert the AST back into source code