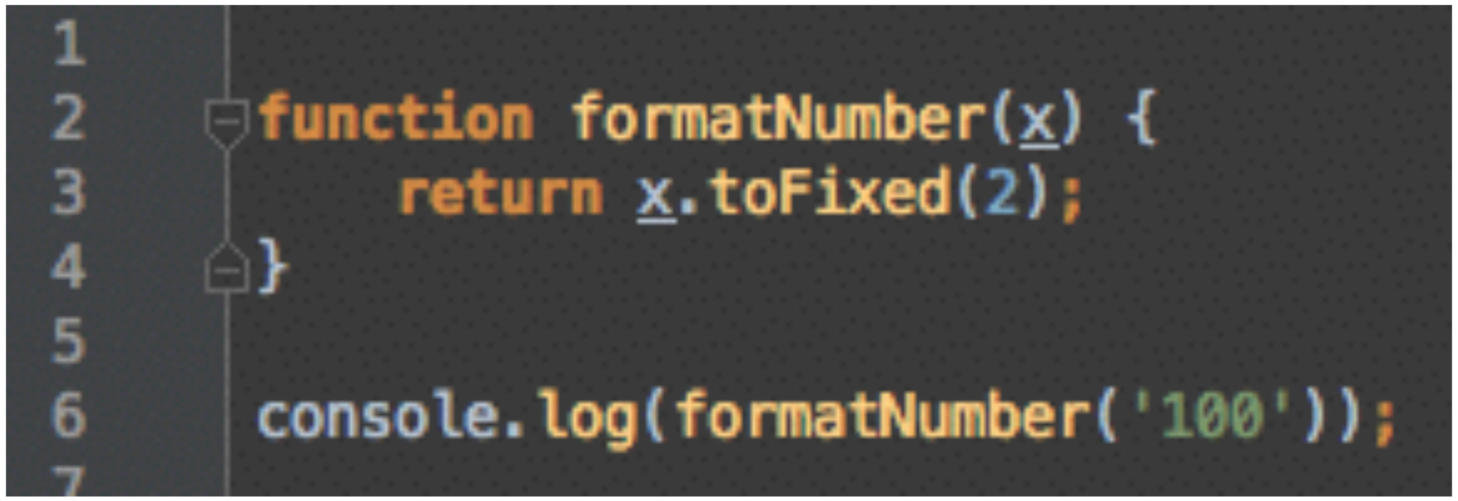
Typed JavaScript

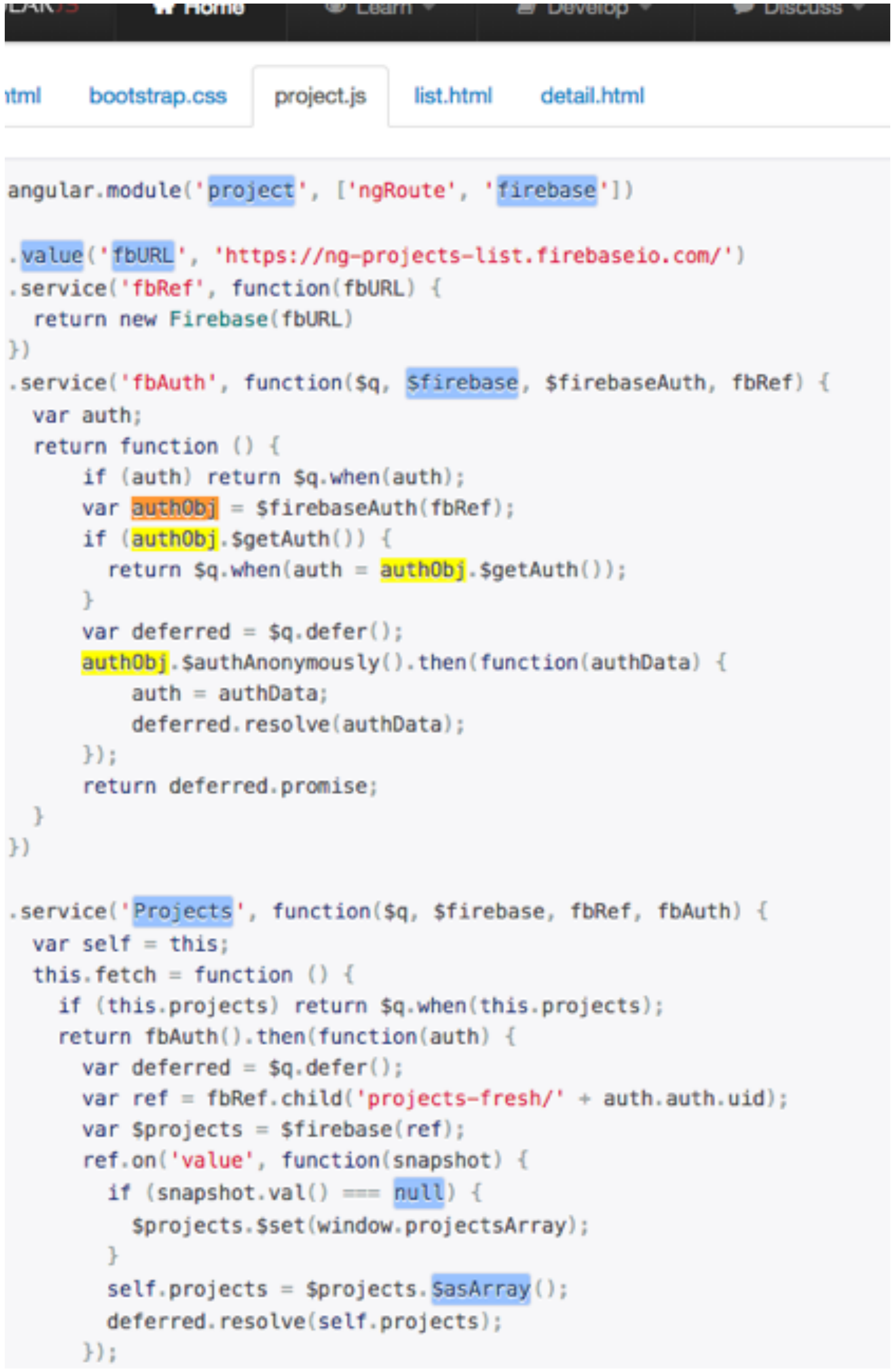
Type check



```

function formatNumber(x) {
    return x.toFixed(2);
}

console.log(formatNumber('100'));
```

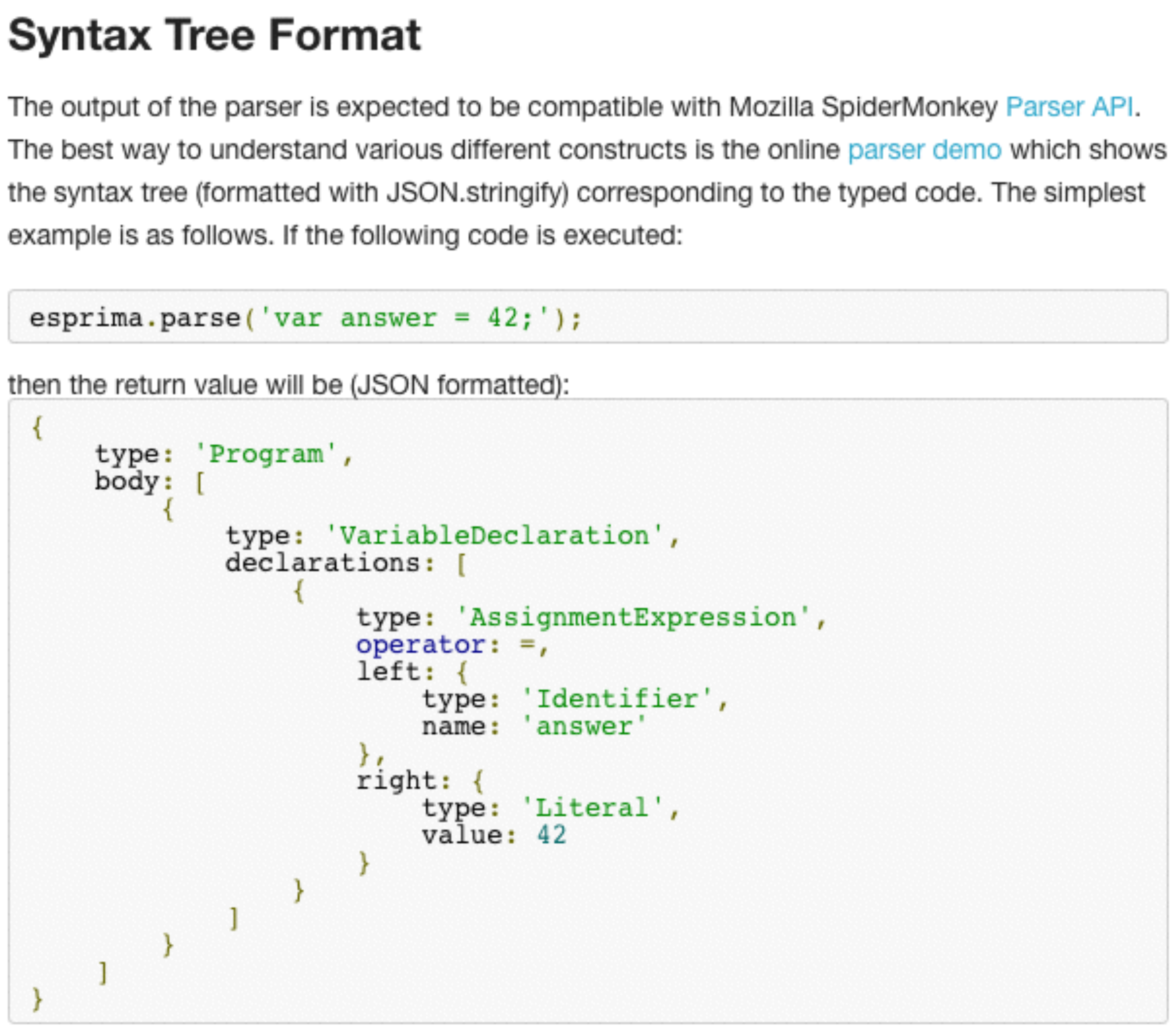
API

AngularJS



```
angular.module('project', ['ngRoute', 'firebase'])

.value('fbURL', 'https://ng-projects-list.firebaseio.com/')
.service('fbRef', function(fbURL) {
  return new Firebase(fbURL)
})
.service('fbAuth', function($q, $firebase, $firebaseAuth, fbRef) {
  var auth;
  return function () {
      if (auth) return $q.when(auth);
      var authObj = $firebaseAuth(fbRef);
      if (authObj.$getAuth()) {
        return $q.when(auth = authObj.$getAuth());
      }
      var deferred = $q.defer();
      authObj.$authAnonymously().then(function(authData) {
          auth = authData;
          deferred.resolve(authData);
      });
      return deferred.promise;
  }
})

.service('Projects', function($q, $firebase, fbRef, fbAuth) {
  var self = this;
  this.fetch = function () {
    if (this.projects) return $q.when(this.projects);
    return fbAuth().then(function(auth) {
      var deferred = $q.defer();
      var ref = fbRef.child('projects-fresh/' + auth.auth.uid);
      var $projects = $firebase(ref);
      ref.on('value', function(snapshot) {
        if (snapshot.val() === null) {
          $projects.$set(window.projectsArray);
        }
        self.projects = $projects.$asArray();
        deferred.resolve(self.projects);
      });
```

Esprima



Syntax Tree Format

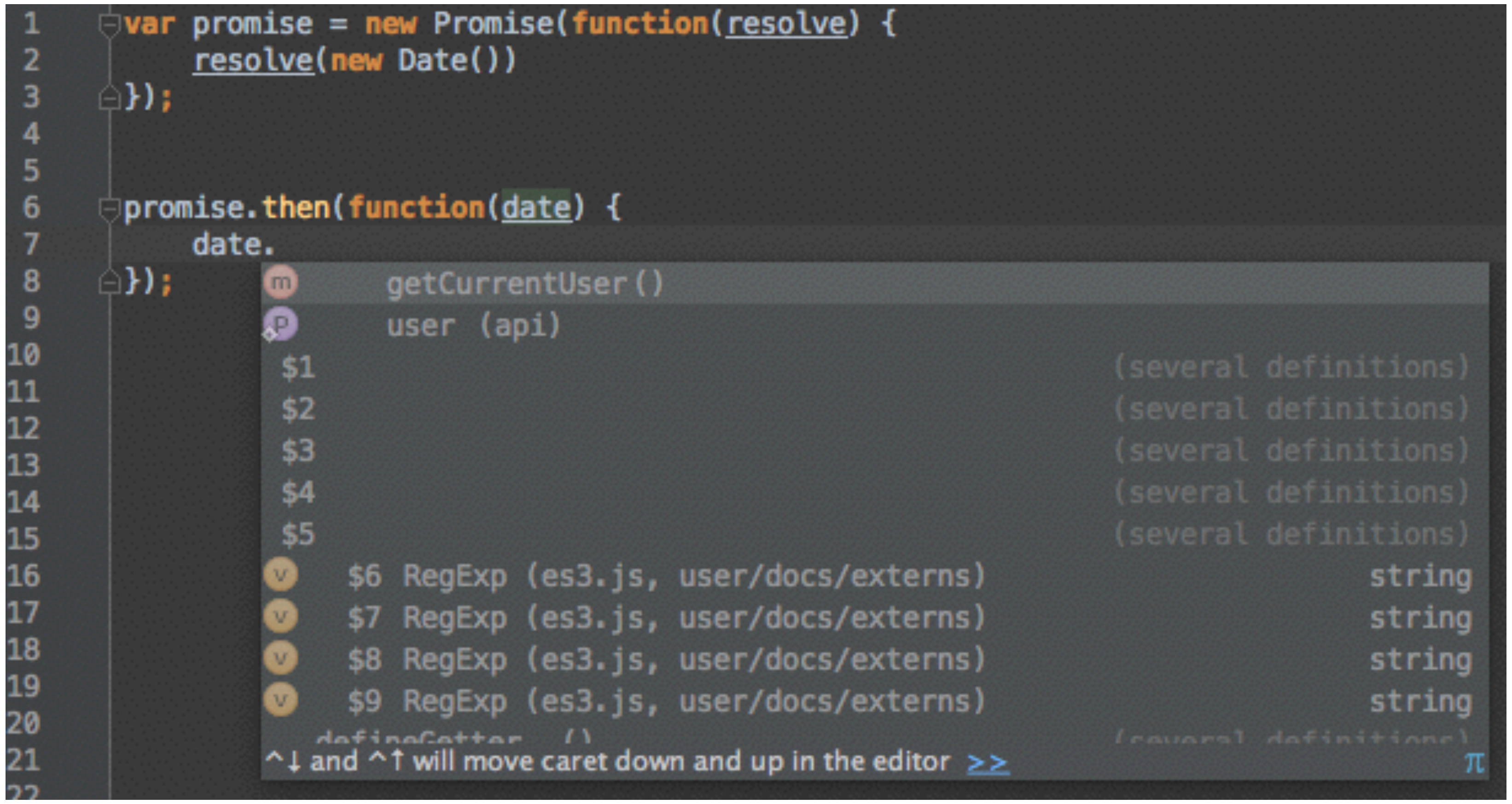
The output of the parser is expected to be compatible with Mozilla SpiderMonkey Parser API. The best way to understand various different constructs is the online parser demo which shows the syntax tree (formatted with JSON.stringify) corresponding to the typed code. The simplest example is as follows. If the following code is executed:

```
esprima.parse('var answer = 42;');
```

then the return value will be (JSON formatted):

```
{
    type: 'Program',
    body: [
        {
            type: 'VariableDeclaration',
            declarations: [
                {
                    type: 'AssignmentExpression',
                    operator: =,
                    left: {
                        type: 'Identifier',
                        name: 'answer'
                    },
                    right: {
                        type: 'Literal',
                        value: 42
                    }
                }
            ]
        }
    ]
}
```

Code Completion

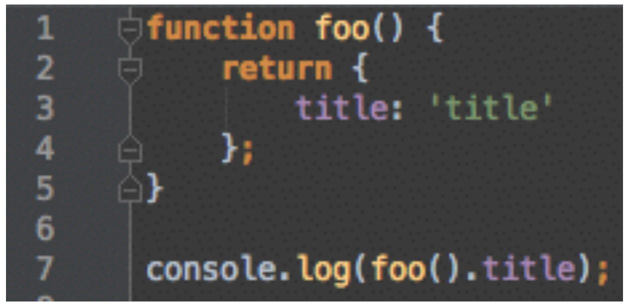


```
var promise = new Promise(function(resolve) {
    resolve(new Date())
});


promise.then(function(date) {
    date.
});
```

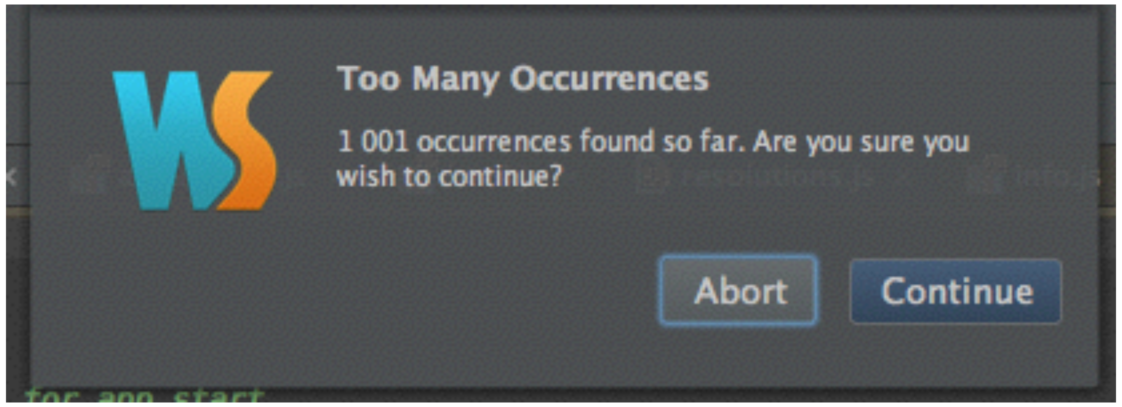
```
getCurrentUser()
user (api)
$1                                                  (several definitions)
$2                                                  (several definitions)
$3                                                  (several definitions)
$4                                                  (several definitions)
$5                                                  (several definitions)
$6 RegExp (es3.js, user/docs/externs)                              string
$7 RegExp (es3.js, user/docs/externs)                              string
$8 RegExp (es3.js, user/docs/externs)                              string
$9 RegExp (es3.js, user/docs/externs)                              string
```

^↓ and ^↑ will move caret down and up in the editor >>

Refactoring



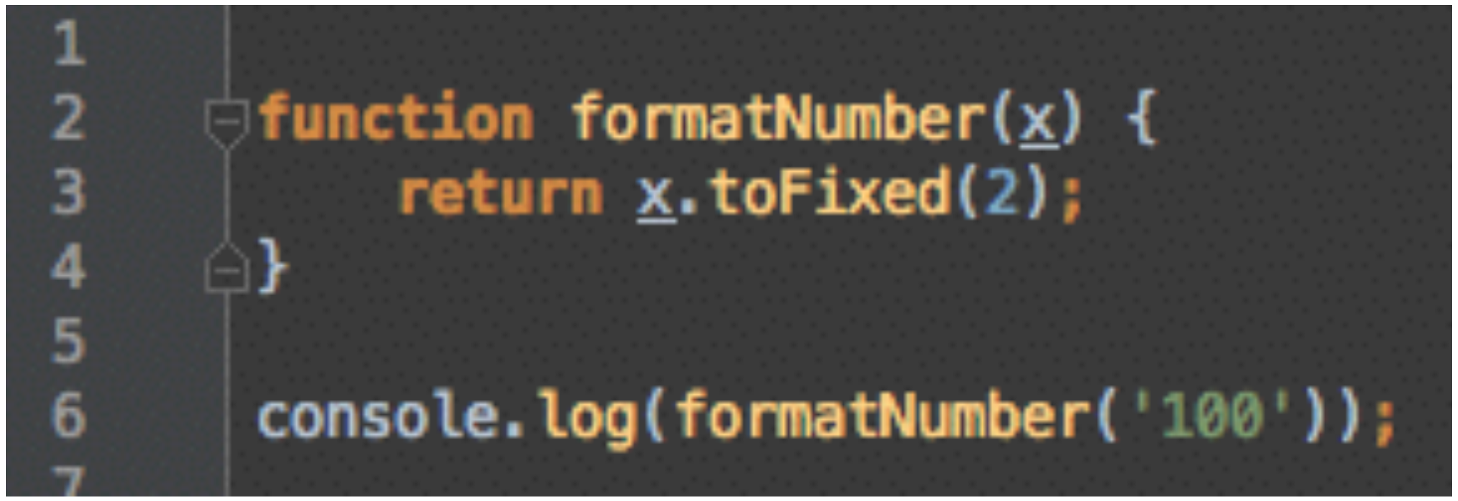
```
function foo() {
    return {
        title: 'title'
    };
}

console.log(foo().title);
```



Too Many Occurrences

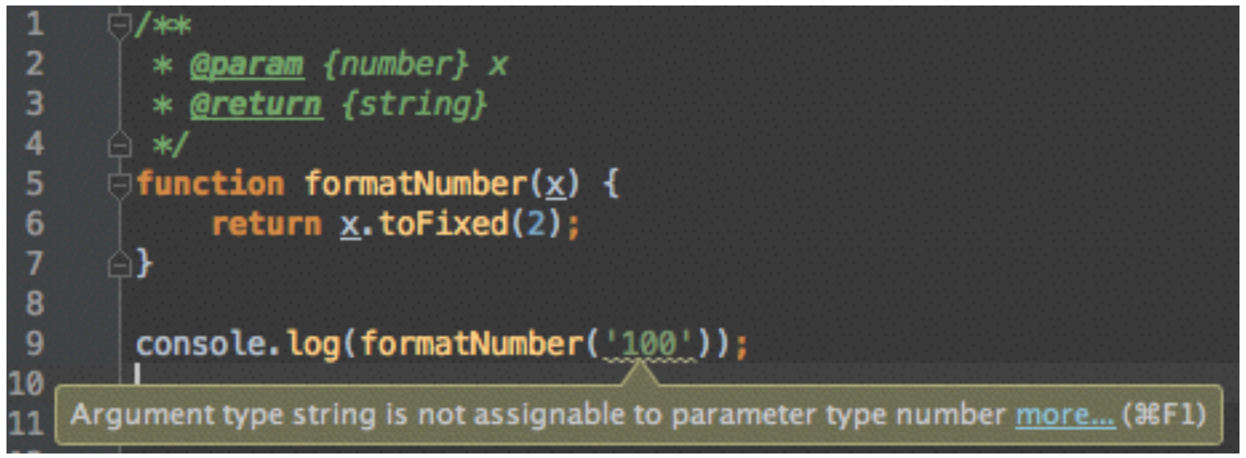
1 001 occurrences found so far. Are you sure you wish to continue?

Type check



```

function formatNumber(x) {
    return x.toFixed(2);
}

console.log(formatNumber('100'));
```

Type check

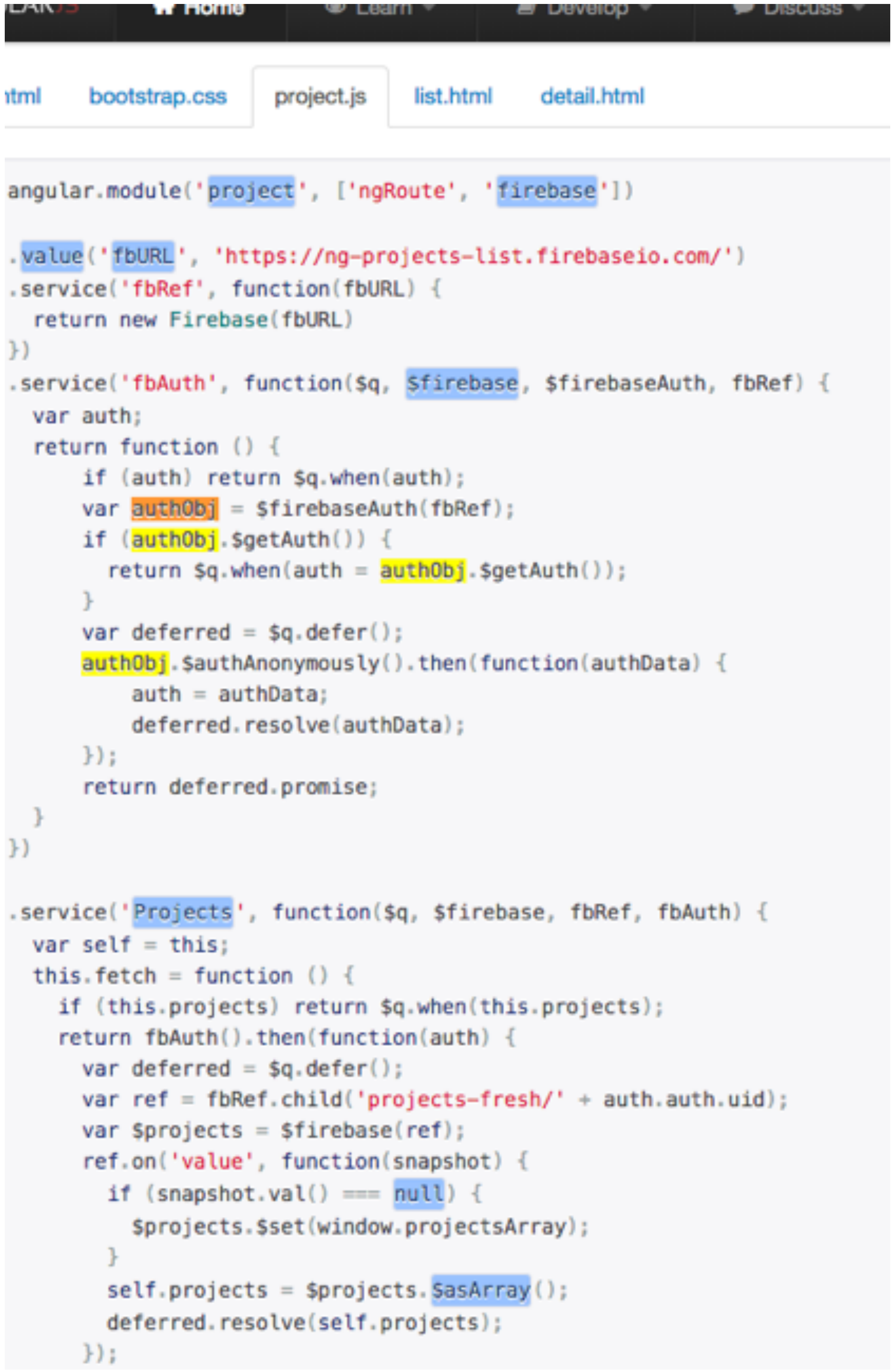


```
/**
 * @param {number} x
 * @return {string}
 */
function formatNumber(x) {
    return x.toFixed(2);
}

console.log(formatNumber('100'));

```

Argument type string is not assignable to parameter type number more... (⌘F1)

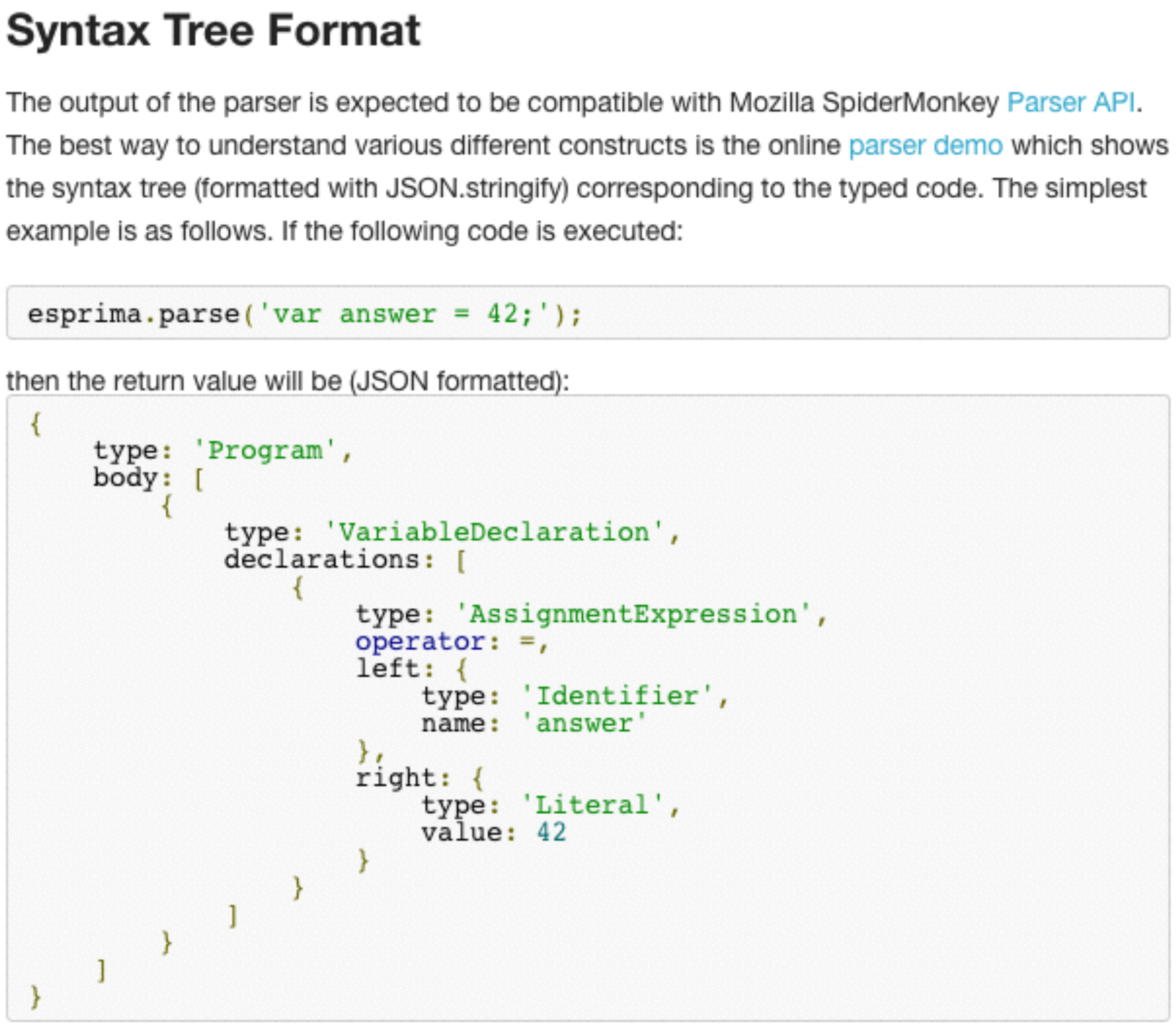
API

AngularJS



```
angular.module('project', ['ngRoute', 'firebase'])

.value('fbURL', 'https://ng-projects-list.firebaseio.com/')
.service('fbRef', function(fbURL) {
  return new Firebase(fbURL)
})
.service('fbAuth', function($q, $firebase, $firebaseAuth, fbRef) {
  var auth;
  return function () {
      if (auth) return $q.when(auth);
      var authObj = $firebaseAuth(fbRef);
      if (authObj.$getAuth()) {
        return $q.when(auth = authObj.$getAuth());
      }
      var deferred = $q.defer();
      authObj.$authAnonymously().then(function(authData) {
          auth = authData;
          deferred.resolve(authData);
      });
      return deferred.promise;
  }
})

.service('Projects', function($q, $firebase, fbRef, fbAuth) {
  var self = this;
  this.fetch = function () {
    if (this.projects) return $q.when(this.projects);
    return fbAuth().then(function(auth) {
      var deferred = $q.defer();
      var ref = fbRef.child('projects-fresh/' + auth.auth.uid);
      var $projects = $firebase(ref);
      ref.on('value', function(snapshot) {
        if (snapshot.val() === null) {
          $projects.$set(window.projectsArray);
        }
        self.projects = $projects.$asArray();
        deferred.resolve(self.projects);
      });
```

Esprima



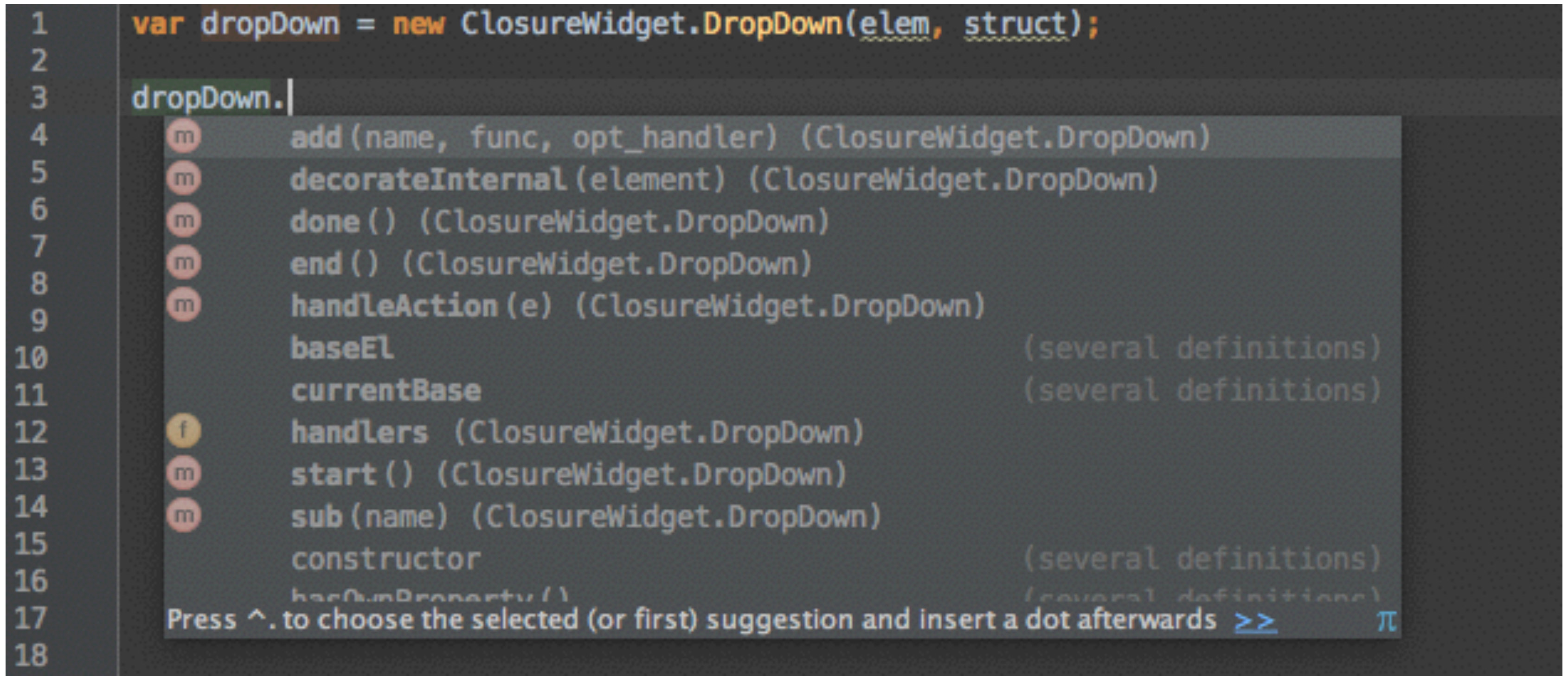
Syntax Tree Format

The output of the parser is expected to be compatible with Mozilla SpiderMonkey Parser API. The best way to understand various different constructs is the online parser demo which shows the syntax tree (formatted with JSON.stringify) corresponding to the typed code. The simplest example is as follows. If the following code is executed:

```
esprima.parse('var answer = 42;');
```

then the return value will be (JSON formatted):

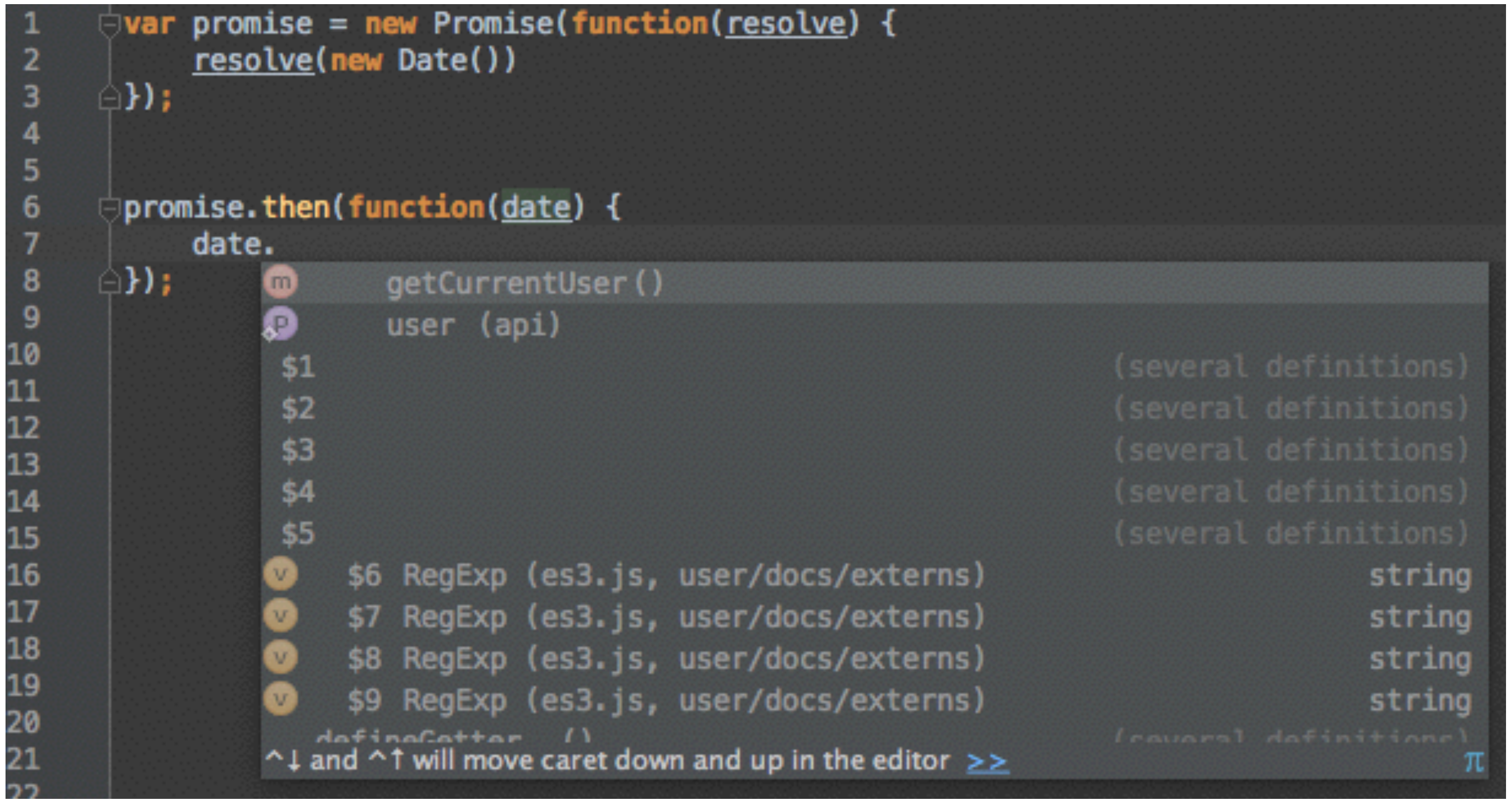
```
{
    type: 'Program',
    body: [
        {
            type: 'VariableDeclaration',
            declarations: [
                {
                    type: 'AssignmentExpression',
                    operator: =,
                    left: {
                        type: 'Identifier',
                        name: 'answer'
                    },
                    right: {
                        type: 'Literal',
                        value: 42
                    }
                }
            ]
        }
    ]
}
```



```
var dropDown = new ClosureWidget.DropDown(elem, struct);

dropDown.
    add(name, func, opt_handler) (ClosureWidget.DropDown)
    decorateInternal(element) (ClosureWidget.DropDown)
    done() (ClosureWidget.DropDown)
    end() (ClosureWidget.DropDown)
    handleAction(e) (ClosureWidget.DropDown)
    baseEl                                  (several definitions)
    currentBase                             (several definitions)
    handlers (ClosureWidget.DropDown)
    start() (ClosureWidget.DropDown)
    sub(name) (ClosureWidget.DropDown)
    constructor                             (several definitions)
    hasOwnProperty()                        (several definitions)
```

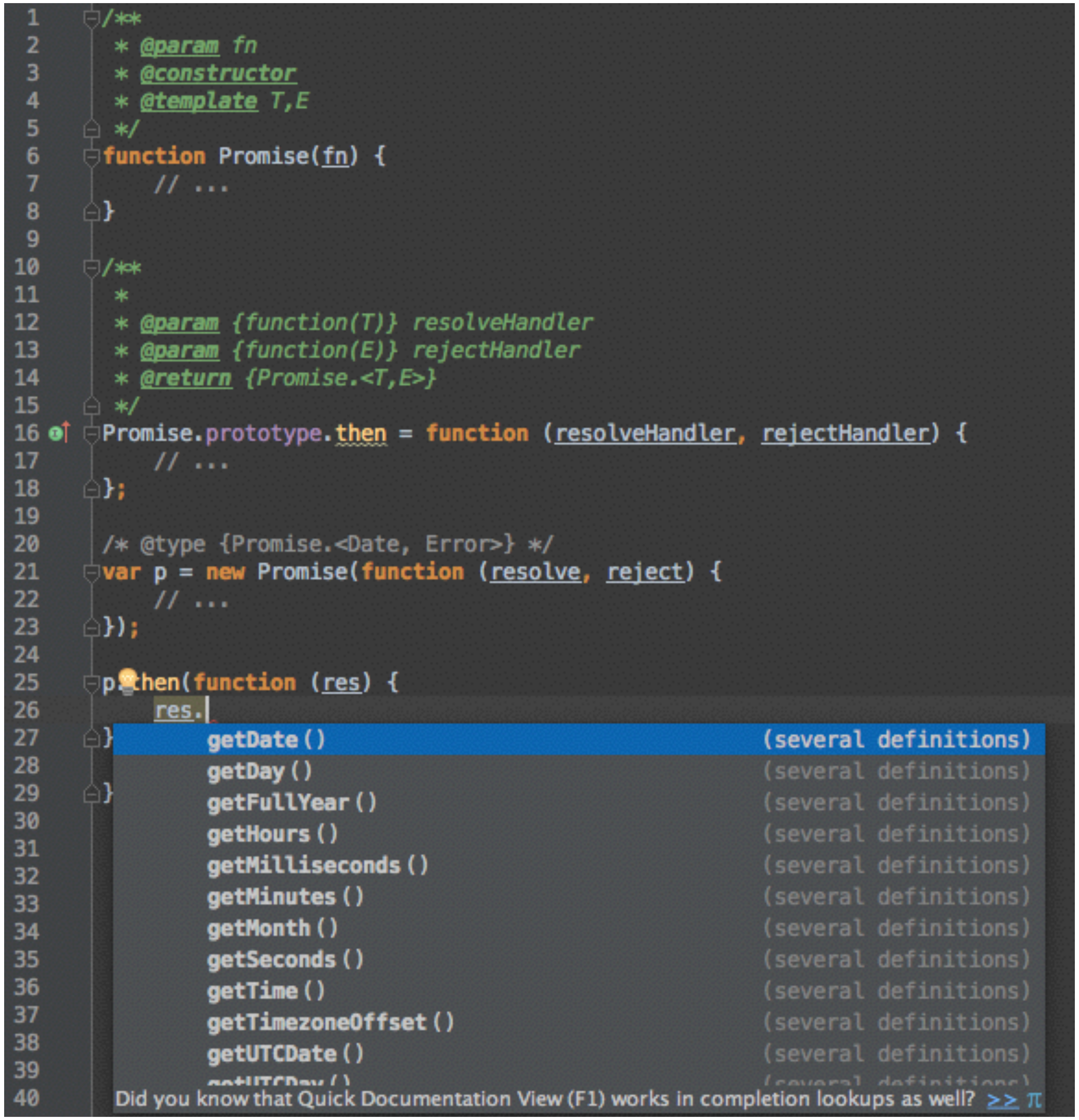
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >>

Code Completion



```
var promise = new Promise(function(resolve) {
    resolve(new Date())
});


promise.then(function(date) {
    date.
});
```

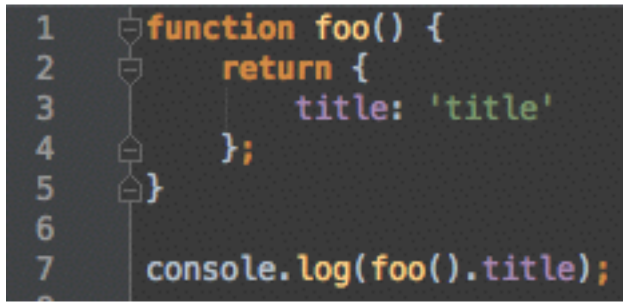
```
getCurrentUser()
user (api)
$1                                              (several definitions)
$2                                              (several definitions)
$3                                              (several definitions)
$4                                              (several definitions)
$5                                              (several definitions)
$6 RegExp (es3.js, user/docs/externs)                          string
$7 RegExp (es3.js, user/docs/externs)                          string
$8 RegExp (es3.js, user/docs/externs)                          string
$9 RegExp (es3.js, user/docs/externs)                          string
defineGetter ()                                 (several definitions)
^↓ and ^↑ will move caret down and up in the editor >>              π
```



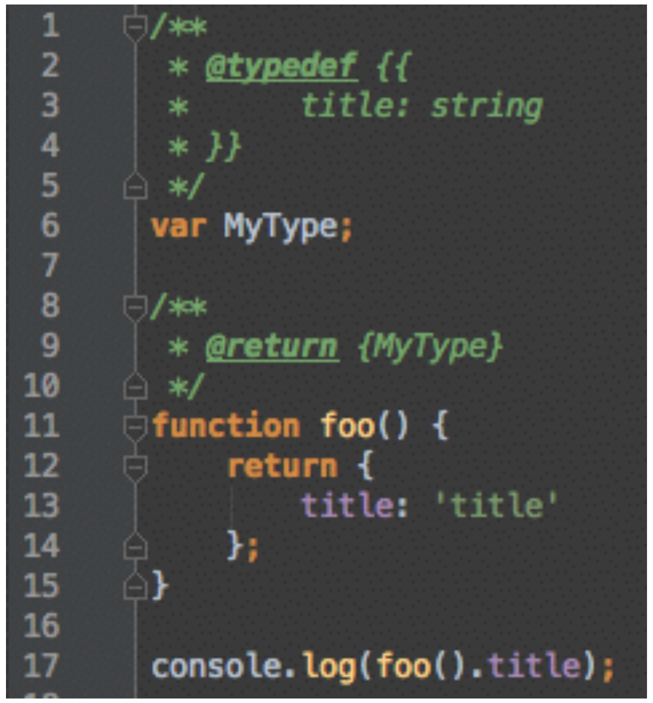
```
/**
 * @param fn
 * @constructor
 * @template T,E
 */
function Promise(fn) {
    // ...
}

/**
 *
 * @param {function(T)} resolveHandler
 * @param {function(E)} rejectHandler
 * @return {Promise.<T,E>}
 */
Promise.prototype.then = function (resolveHandler, rejectHandler) {
    // ...
};

/* @type {Promise.<Date, Error>} */
var p = new Promise(function (resolve, reject) {
    // ...
});

p.then(function (res) {
    res.
}

}
```

```
getDate()                                  (several definitions)
getDay()                                   (several definitions)
getFullYear()                              (several definitions)
getHours()                                 (several definitions)
getMilliseconds()                          (several definitions)
getMinutes()                               (several definitions)
getMonth()                                 (several definitions)
getSeconds()                               (several definitions)
getTime()                                  (several definitions)
getTimezoneOffset()                        (several definitions)
getUTCDate()                               (several definitions)
getUTCDay()                                (several definitions)
Did you know that Quick Documentation View (F1) works in completion lookups as well? >> π
```

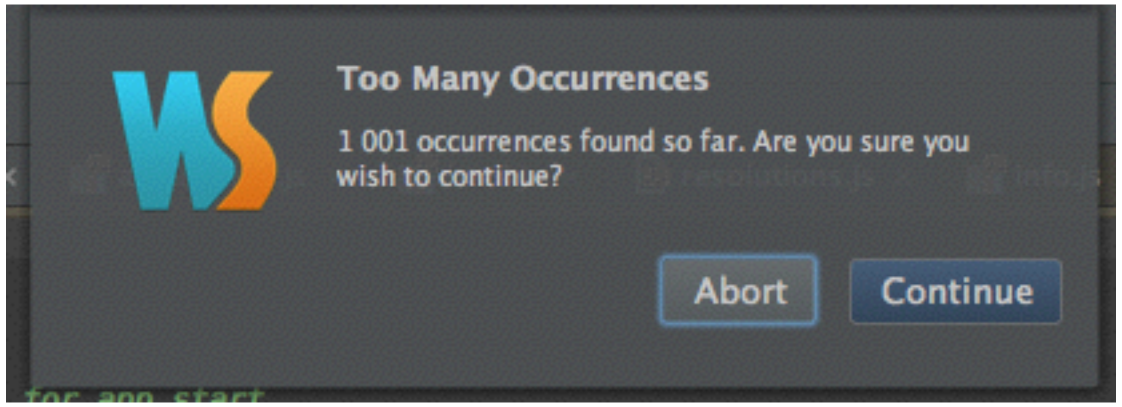
Refactoring



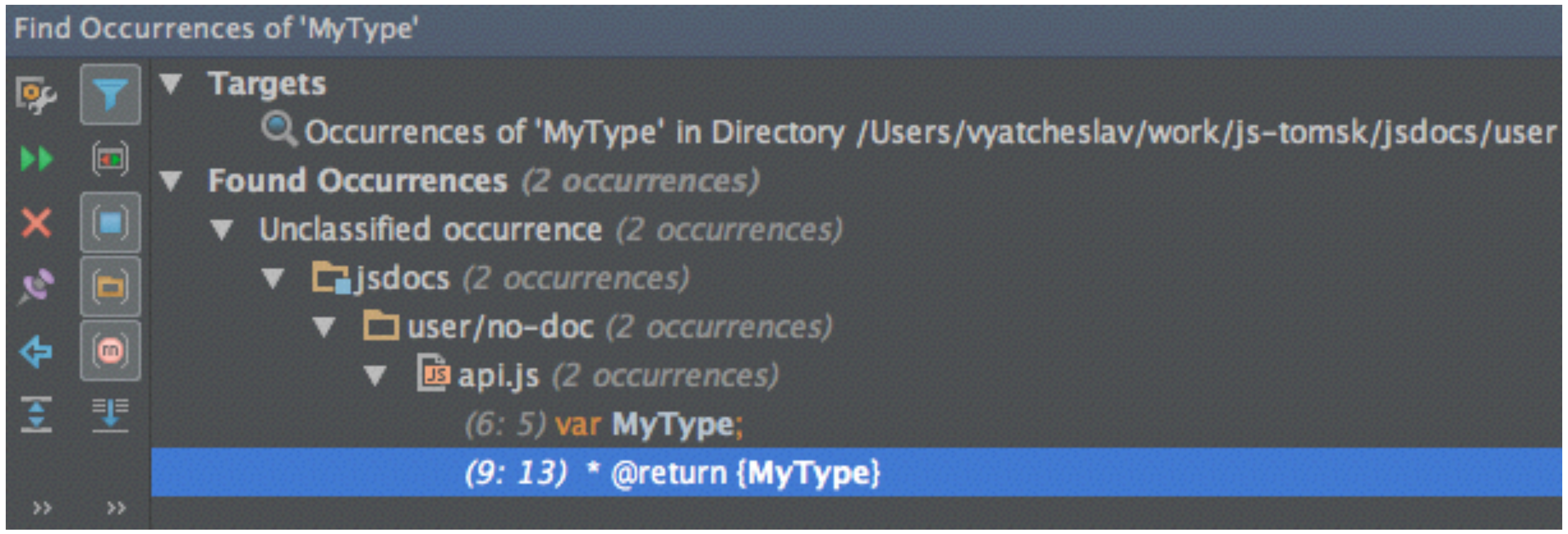
```
function foo() {
    return {
        title: 'title'
    };
}

console.log(foo().title);
```



```
/**
 * @typedef {{
 *      title: string
 * }}
 */
var MyType;

/**
 * @return {MyType}
 */
function foo() {
    return {
        title: 'title'
    };
}

console.log(foo().title);
```


Too Many Occurrences

1 001 occurrences found so far. Are you sure you wish to continue?



```
Find Occurrences of 'MyType'
▼ Targets
    Occurrences of 'MyType' in Directory /Users/vyatcheslav/work/js-tomsk/jsdocs/user
▼ Found Occurrences (2 occurrences)
  ▼ Unclassified occurrence (2 occurrences)
    ▼ jsdocs (2 occurrences)
      ▼ user/no-doc (2 occurrences)
        ▼ api.js (2 occurrences)
            (6: 5) var MyType;
            (9: 13) * @return {MyType}
```

- Microsoft TypeScript - typed ES superset
- Google AtScript - ES based typed language
- Facebook Flow - static type checker
- Google Closure Compiler - static type checker, compiler, uglifier, compressor, etc.
- JSDoc - annotations

Вячеслав Зайцев

slava@interfaced.ru

Interfaced.