ECMASCRIPT 2015

ES6

~~var~~, const, let

```
1 var v = 1;
2 const a = 1;
3 let b = 1;
4
5 v = 2; // OK
6 a = 2; // TypeError: Assignment to constant variable.
7 b = 2; // OK
8
```

```
function f() {
  console.log(a); // OK
  console.log(b); // ReferenceError: b is not defined

  var a = 1;
  let b = 1;
}

f();
```

Arrow Functions

```
1 var oldF = function(number) {
2   return number % 2 === 0;
3 };
4
5 const newF = (number) => {
6   return number % 2 === 0;
7 }
8
9 const newF2 = (number) => number % 2 === 0;
```

```
const numbers = [1, 2, 3, 4, 5, 6];

const evenNumbers = numbers.filter(function(num) {
  return num % 2 === 0;
});

console.log(evenNumbers); // [ 2, 4, 6 ]

```

```
const evenNumbers = numbers.filter(num => num % 2 === 0);
```

```
const f = () => {
  return {
    name: 'John',
    age: 42,
  };
}


const f = () => ({ name: 'John', age: 42 });
```

```
var obj = {
  name: 'obj',
  items: [1, 2],
  f: function() {
    console.log(this.name); // obj
    this.items.forEach(function(item) {
      console.log(this.name); // undefined, undefined
    });
  },
};

obj.f();
```

```
var obj2 = {
  name: 'obj',
  items: [1, 2],
  f: function() {
    console.log(this.name); // obj
    this.items.forEach((item) => {
      console.log(this.name); // obj, obj
    });
  },
};

obj2.f();
```

Default Parameters

```
1 const f = (a, b, c) => {
2   b = b || 2;
3   c = c || 3;
4
5   console.log(a, b, c);
6 }
7
8 f(1); // 1, 2, 3
```

```
1 const f = (a, b = 2, c = 3) => {
2   console.log(a, b, c);
3 }
4
```

Rest Parameters

```
1 const f = (a, b, ...other) => {
2   console.log(a); // 1
3   console.log(b); // 'hello'
4   console.log(other.length); // 3
5   console.log(other); // [ 0, 'world', 2 ]
6 };
7
8 f(1, 'hello', 0, 'world', 2);
```

Spread Operator

```
1 const params = ["hello", 3];
2 const other = [1, 2, ...params]; // [ 1, 2, "hello", 3 ]
3 // equals:
4 const other = [1, 2, 'hello', 3];
```

```
1 const params = ["hello", 3];
2
3 const f = (a, b, c, d, e) => {
4   console.log(a, b, c, d, e);
5   // 1 2 'hello' 3 undefined
6 }
7
8 f(1, 2, ...params);
```

Template Literals

```
1 const name = 'John';
2 const a = `Hello, ${name}!`; // Hello, John!
3
4 const number = 2;
5 const b = `Result: ${number * 2 + 2}`; // Result: 6
```

Property Shorthand

```
1  const name = 'John';
2  const age = 42;
3
4  const person = {
5    name: name,
6    age: age,
7  };
8
9  const goodPerson = {
10   name,
11   age
12 };
```

Destructuring

```
1  const f = () => {
2    return {
3      name: 'John',
4      age: 42,
5    };
6  }
7
8  console.log(f()); // { name: 'John', age: 42 }
9
10 const { name, age } = f();
11
12 console.log(name); // John
13 console.log(age); // 42
14
```

```
const f = () => {
  return {
    info: {
      firstName: 'John',
      lastName: 'Brown',
    },
    age: 42,
    address: {
      city: 'San Francisco',
      state: 'CA',
    }
  };
}

const { info: { firstName }, address: { city }, age } = f();

console.log(`${firstName} (${age} y.o.) from ${city}`);
// John (42 y.o.) from San Francisco

console.log(info); // ReferenceError: info is not defined
console.log(address); // ReferenceError: address is not defined
```

```
const person = {
  info: {
    firstName: 'John',
    lastName: 'Brown',
  },
  age: 42,
  address: {
      city: 'San Francisco',
      state: 'CA',
    }
};

const print = (data) => {
  console.log(`${data.info.firstName} (${data.age} y.o.) ` +
    `from ${data.address.city}`);
};

print(person);
```

```
const person = {
  info: {
    firstName: 'John',
    lastName: 'Brown',
  },
  age: 42,
  address: {
      city: 'San Francisco',
      state: 'CA',
    }
};

const print = ({ info: { firstName }, address: { city }, age }) => {
  console.log(`${firstName} (${age} y.o.) from ${city}`);
};

print(person);
```

Modules

```
1 export const PI = 3.14;
2 export const f = () => "Works";
3 export default () => "Default";
4
```

```
1 import * as myModule from './module.js';
2
3 console.log(myModule.PI);
4 console.log(myModule.f());
```

```
1 import myFunc, { PI, f } from './module.js';
2
3 console.log(PI);
4 console.log(f());
5 console.log(myFunc());
6
```

Classes

```
1  class Shape {
2    constructor (id, x, y) {
3      this.id = id
4      this.move(x, y)
5    }
6    move (x, y) {
7      this.x = x
8      this.y = y
9    }
10
11   toString() {
12     return `${this.id} (x: ${this.x}, y: ${this.y})`;
13   }
14 }
15
16 const shape = new Shape('s1', 3, 3);
17 console.log(shape.toString()); // s1 (x: 3, y: 3)
```

```
class Circle extends Shape {
  constructor (id, x, y, radius) {
    super(id, x, y)
    this.radius = radius
  }

  toString() {
    return `[Circle radius: ${this.radius}] ${super.toString()}`;
  }
}

const c = new Circle('s1', 3, 3, 10);
console.log(c.toString()); // [Circle radius: 10] s1 (x: 3, y: 3)
```

```
class Circle extends Shape {
  constructor (id, x, y, radius) {
    super(id, x, y)
    this.radius = radius
  }

  static defaultCircle () {
    return new Circle("default", 0, 0, 100)
  }

  toString() {
    return `[Circle radius: ${this.radius}] ${super.toString()}`;
  }
}

const c = Circle.defaultCircle();
console.log(c.toString()); // [Circle radius: 100] default (x: 0, y: 0)
```

Generators

```
function* generator() {
  yield 1;
  yield 2;
  console.log(yield 3);
}

const gen = generator();
console.log(gen.next()); // { value: 1, done: false }
console.log(gen.next()); // { value: 2, done: false }
console.log(gen.next()); // { value: 3, done: false }
console.log(gen.next(3)); // { value: undefined, done: true }



function bar(x) {
  console.log("x: " + x);
}

function *fooGenerator() {
  yield;
  bar(yield);
}

const foo = fooGenerator()

console.log(foo.next()); // { value: undefined, done: false }
console.log(foo.next()); // { value: undefined, done: false }
console.log(foo.next(2)); // { value: undefined, done: true }
```

```
function* range(start, end, step) {
  while (start < end) {
    yield start;
    start += step;
  }
}

for (let i of range(0, 10, 2)) {
  console.log(i); // 0, 2, 4, 6, 8
}
```

New Built-In Methods

```
1  const a = { key: 'value' };
2  const b = Object.assign({}, a);
3  console.log(b); // { key: 'value' }
4  console.log(a == b); // false
5
6  const c = Object.assign({}, a, { hello: 'world' });
7  console.log(c); // { key: 'value', hello: 'world ' }
8
9  const d = Object.assign(a, { hello: 'world' });
10 console.log(d); // { key: 'value', hello: 'world ' }
11 console.log(a); // { key: 'value', hello: 'world ' }
12 console.log(d == a); // true
```

```
1 [ 1, 3, 4, 2 ].find(x => x > 3) // 4
2
3 "hello".startsWith("ello", 1) // true
4 "hello".endsWith("hell", 4)   // true
5 "hello".includes("ell")       // true
6 "hello".includes("ell", 1)    // true
7 "hello".includes("ell", 2)    // false
8
```

Promises

Promises are a first class representation of a value that may be made available in the future

```
new Promise((resolve, reject) => {
  resolve('value');
})
```

```
const f = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => reject(new Error(':(')), 1000);
  });
};

f()
.then((result) => {
  console.log('Promise resolved', result);
})
.catch((err) => {
  console.log(err);
});
```

```
asyncOperation((data) => {
  anotherAsync((data2) => {
    yetAnotherAsync(() => {
      // Yay we're finished!
    });
  });
});
```

```
asyncOperation(handler1);

function handler1(data) {
    // Do some processing with `data`
    anotherAsync(handler2);
}

function handler2(data2) {
    // Some more processing with `data2`
    yetAnotherAsync(handler3);
}

function handler3() {
    // Yay we're finished!
}
```

```
13 asyncOperation()
14   .then((data) => {
15     // Do some processing with `data`
16     return anotherAsync();
17   })
18   .then((data2) => {
19     // Some more processing with `data2`
20     return yetAnotherAsync();
21   })
22   .then(() => {
23     // Yay we're finished!
24   });
25
```

```
const get = (url) => {
  return new Promise((resolve, reject) => {
    const req = new XMLHttpRequest();
    req.open('GET', url);

    req.onload = () => {
      if (req.status === 200) {
        resolve(req.response);
      } else {
        reject(Error(req.statusText));
      }
    };

    req.onerror = () => {
      reject(Error('Network Error'));
    };

    req.send();
  });
};

get('story.json')
  .then((response) => {
    console.log('Success!', response);
  })
  .catch((error) => {
    console.error('Failed!', error);
  });
```

```
const f = () => {
  return new Promise((resolve, reject) => {
    resolve('value 1');
  });
};

const g = () => {
  return new Promise((resolve, reject) => {
    resolve('value 2');
  });
};

const result = f()
  .then((value) => {
    console.log(value); // value 1
    return g();
  })
  .then((value) => {
    console.log(value); // value 2
    return value;
  })

console.log(result); // Promise { <pending> }

```

```
fs.readFile('/etc/passwd', (err, data) => {
  if (err) {
    return done(err);
  }

  db.saveData(data, (err) => {
    if (err) {
      return done(err);
    }
    done();
  })
});
```

```
fs.readFile('/etc/passwd')
  .then((data) => {
    return db.saveData();
  })
  .then(done) // .then(() => done())
  .catch(handleError);
```

```
const findUser = (id) => {
  return new Promise((resolve) => {
    setTimeout(() => resolve({ id }), 1000);
  })
}

const findJohn = findUser('john');
const findAlex = findUser('alex');

Promise.all([findJohn, findAlex])
.then((result) => {
  console.log(result);
  // [ { id: 'john' }, { id: 'alex' } ]
});
```

```
Promise.race([findJohn, findAlex])
.then((result) => {
  console.log(result);
  // { id: 'john' }
});
```

Babel

https://babeljs.io

BABEL Learn ES2015 Setup Plugins Usage Try it out Discuss Chat Issues Blog Twitter GitHub

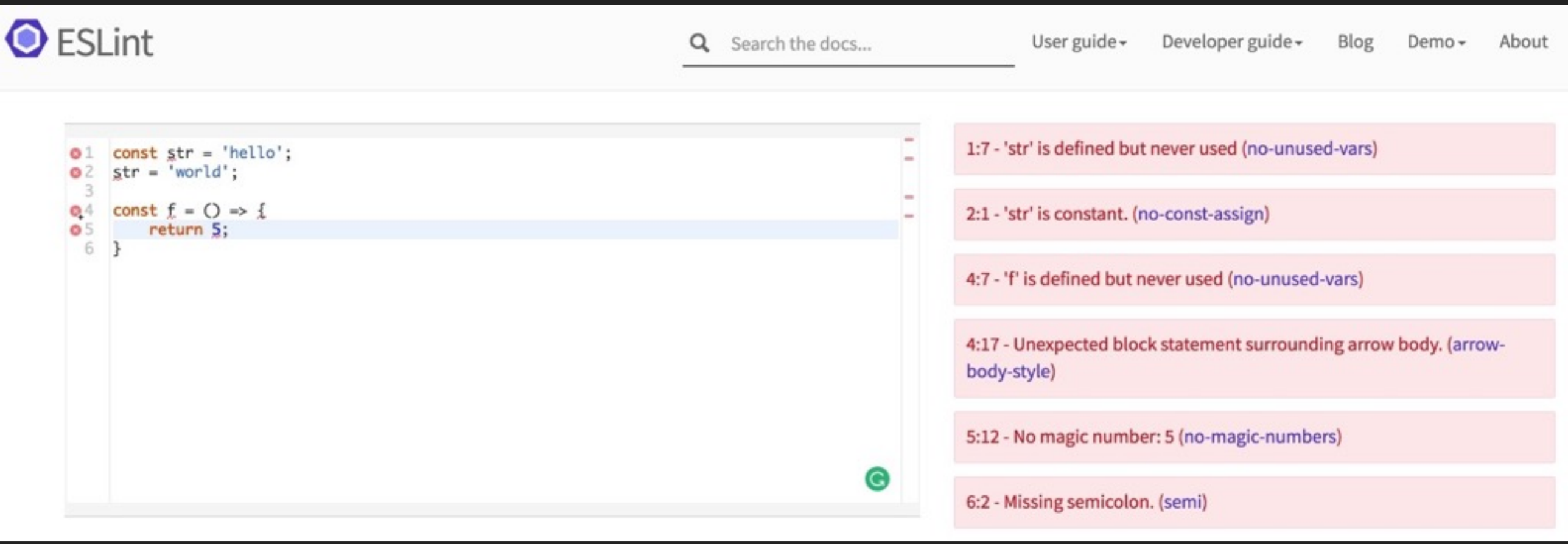
Evaluate Presets: es2015 es2015-loose react stage-0 stage-1 stage-2 stage-3 Line Wrap Babel 6.9.1

```
1  import config from '../../config';
2  import socketTypes from '../../common/socketTypes';
3  import socketWrapper from '../socketWrapper';
4
5  const io = socketWrapper.emitter({ host: config.redis.host, port: config.redis.port });
6
7  export default class Sockets {
8    newAugur({ accountId, augur }) {
9      io.broadcast.to(`account_${accountId}`).emit(socketTypes.augur.created(), augur);
10   }
11 }
12
```

```
9  var _config = require('../../config');
10
11 var _config2 = _interopRequireDefault(_config);
12
13 var _socketTypes = require('../../common/socketTypes');
14
15 var _socketTypes2 = _interopRequireDefault(_socketTypes);
16
17 var _socketWrapper = require('../socketWrapper');
18
19 var _socketWrapper2 = _interopRequireDefault(_socketWrapper);
20
21 function _interopRequireDefault(obj) { return obj && obj.__esModule ? obj : { default: obj }
22
23 function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) {
24
25 var io = _socketWrapper2.default.emitter({ host: _config2.default.redis.host, port: _config2
26
27 var Sockets = function () {
28   function Sockets() {
29     _classCallCheck(this, Sockets);
30   }
31
32   _createClass(Sockets, [{
33     key: 'newAugur',
34     value: function newAugur(_ref) {
35       var accountId = _ref.accountId;
36       var augur = _ref.augur;
37
38       io.broadcast.to('account_' + accountId).emit(_socketTypes2.default.augur.created(), au
39     }
40   }]);
41
42   return Sockets;
43 }();
44
45 exports.default = Sockets;
```

eslint

http://eslint.org/ https://github.com/airbnb/javascript



NEXT GENERATION WEB FRAMEWORK FOR NODE.JS

KOA