

ФГБОУ ВО «Пермский национальный  
исследовательский государственный университет»

Физический факультет  
Кафедра компьютерных систем и телекоммуникаций

# **Классы и наследование, транспиляция в Babel и TypeScript.**

Работу выполнила студентка 2 курса

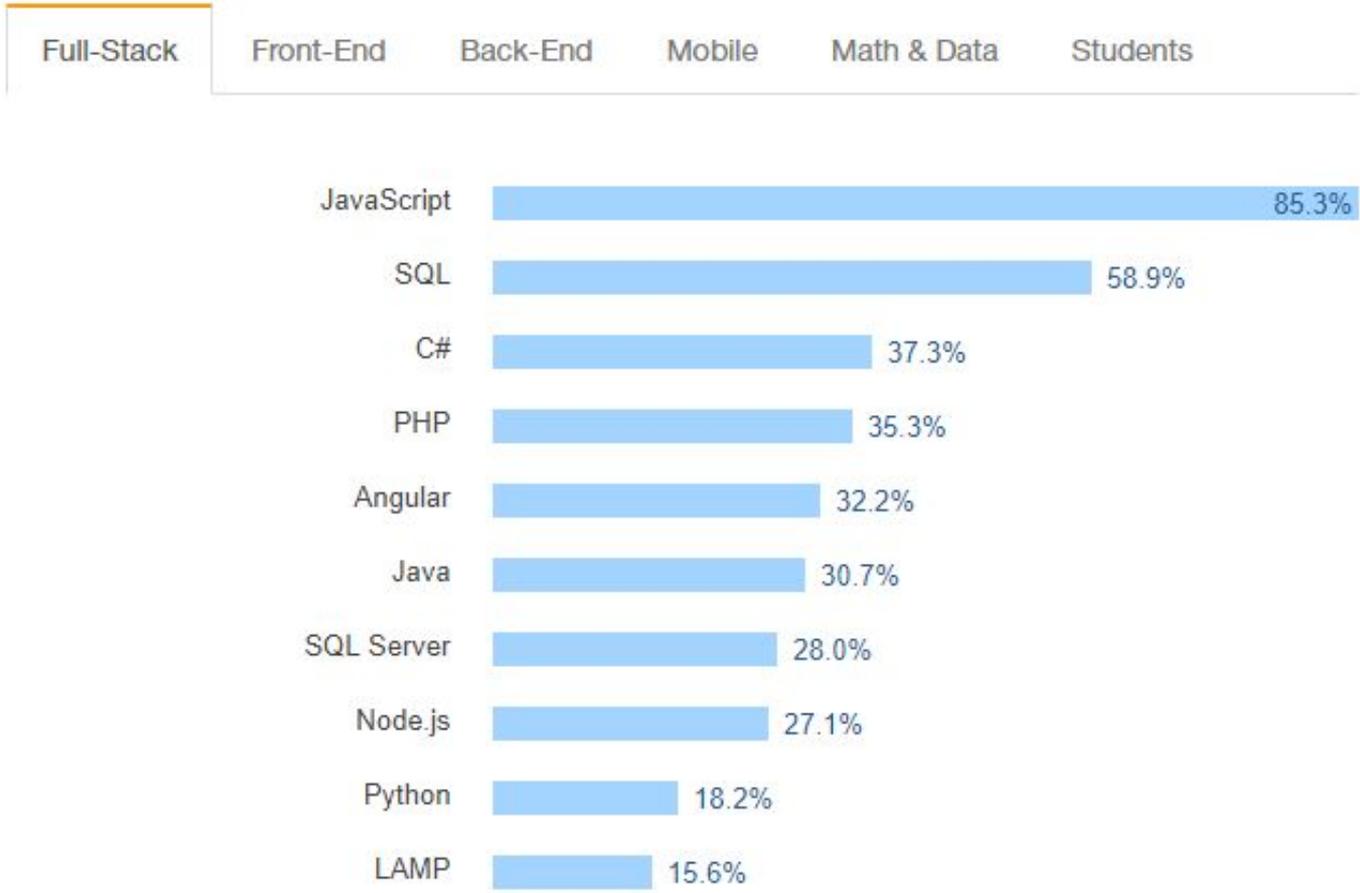
Направление «Радиофизика»  
Профиль магистратуры «Информационные процессы и системы»  
Дегтянникова Дарья Александровна

Пермь, 2019

# Почему JavaScript?

- большое комьюнити;
- мало аналогов;
- много примеров и уже готовых решений;
- быстрый темп роста популярности вэба;
- низкий порог вхождения(справится даже не специалист);
- много библиотек;
- нет нужды в компиляторе(для старта достаточно блокнота и браузера).

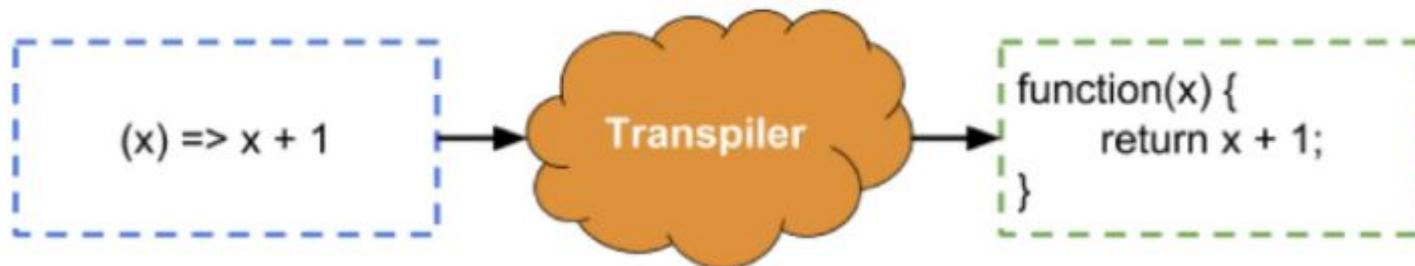
## Most Popular Technologies per Dev Type



<https://insights.stackoverflow.com/survey/2016>

# Понятие транспиляции

- Транспиляция - процесс преобразования кода, написанного с помощью следующих версий языка или на диалектах JavaScript в некий стандартный вариант, понимаемый всеми браузерами.
- Транспиляция - компиляция из одного языка в другой язык с таким же набором абстракций, либо в другую версию того же языка.
- **ECMAScript** — это встраиваемый расширяемый не имеющий средств ввода-вывода язык программирования, используемый в качестве основы для построения других скриптовых языков
- Зачем:
  - Использование ECMAScript6(ES6)
  - Желание писать для вэба не на JS



# Babel(6to5)

Запуск кода на платформе, не поддерживающей новый синтаксис, приведет к синтаксической ошибке. Закономерным решением этой проблемы стало появление Babel — программы, которая берет указанный код и возвращает тот же код, но транслированный в старую версию JS.

## **Состоит из двух частей:**

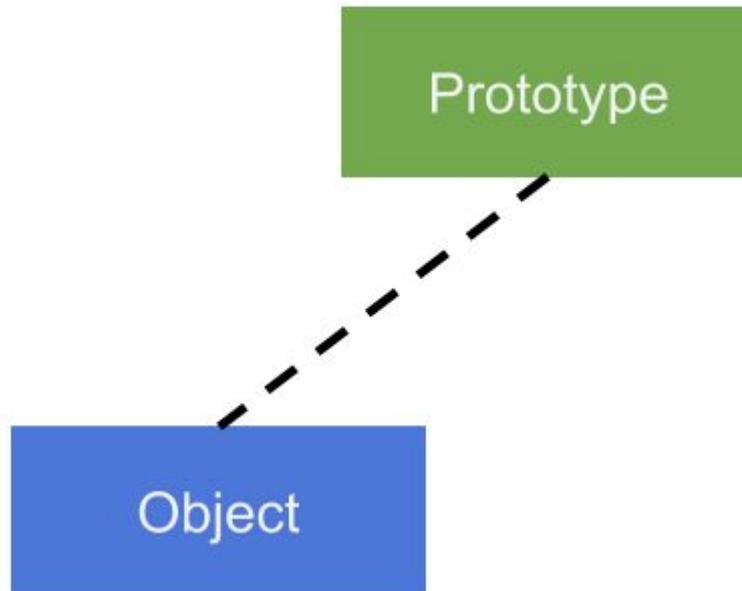
1. Транспайлер, который переписывает код.
2. Полифилл, который добавляет методы.

«Полифилл» (англ. polyfill) – это библиотека, которая добавляет в старые браузеры поддержку возможностей, которые в современных браузерах являются встроенными. Работает она через модификацию стандартных объектов и их прототипов.

# Прототипы объектов

В JavaScript не используется традиционная система наследования, основанная на классах. Для решения схожих задач используются прототипы.

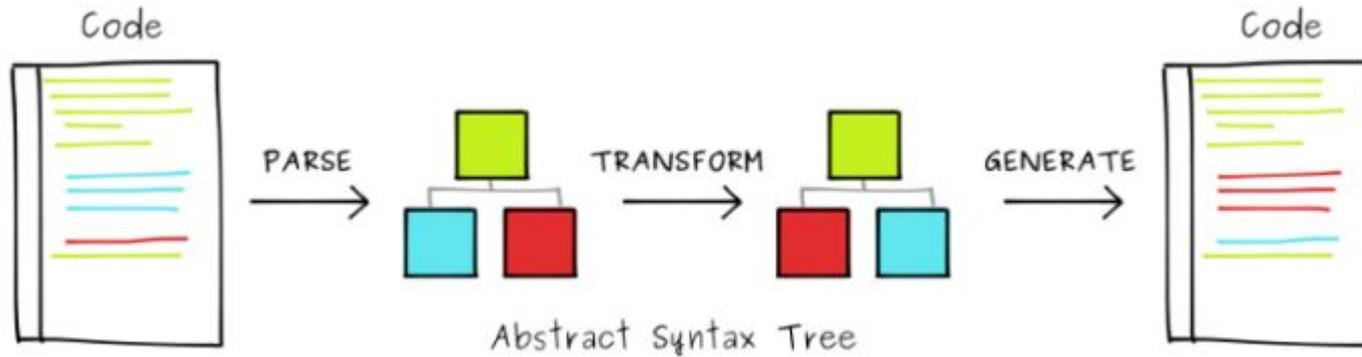
Каждый объект связан с ещё одним объектом — со своим прототипом. Когда вы пытаетесь обратиться к свойству или методу объекта, поиск того, что вам нужно, сначала выполняется в самом объекте. Если поиск не увенчался успехом, он продолжается в прототипе объекта.



# Babel(6to5)

## Принцип действия:

1. Загрузка JS на новом стандарте или на старом стандарте со странными спецификациями;
2. Вызов транспилятора (передача параметров);
3. Получение JS другого стандарта(чаще ES5).



# Лексический анализ

На шаге лексического анализа происходит разбор текста на логические сущности – токены:

- числа;
- строк;
- идентификаторы;
- строки и т.д.

```
function length({ x, y }) {  
    return Math.sqrt(x ** 2 + y ** 2);  
}
```

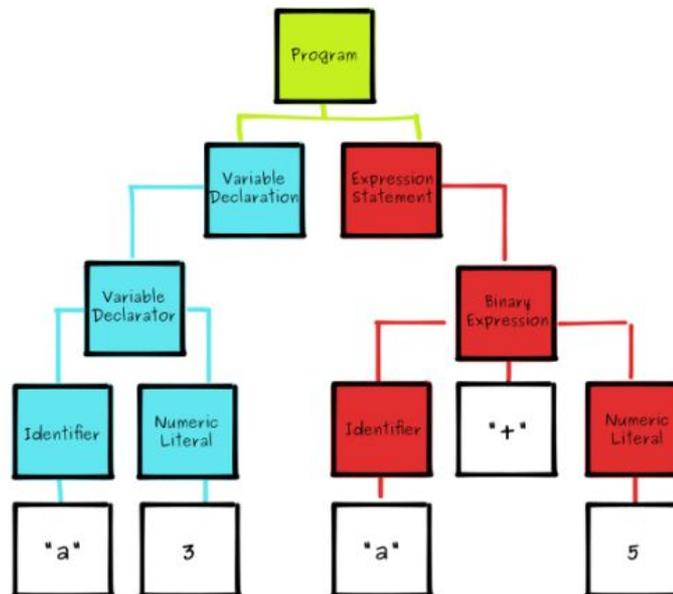


```
function  
length  
(  
{  
x  
,  
y  
...
```

# Синтаксический анализ

На шаге синтаксического анализа происходит создание абстрактного синтаксического дерева (AST) из потока токенов.

```
var a = 3;  
a + 5
```



<https://astexplorer.net/>

# Синтаксический анализ

Math  
.  
sqrt  
(  
x  
\*\*  
2  
+  
y  
\*\*  
2  
)



```
- argument: CallExpression {  
  - callee: MemberExpression {  
    - object: Identifier {  
      name: "Math"  
    }  
    - property: Identifier {  
      name: "sqrt"  
    }  
    computed: false  
  }  
  - arguments: [  
    - BinaryExpression = $node {  
      + left: BinaryExpression {left, operator, right}  
      operator: "+"  
      + right: BinaryExpression {left, operator, right}  
    }  
  ]  
}
```

# Синтаксический анализ

x  
\*\*  
2



```
- left: BinaryExpression {  
  - left: Identifier = $node {  
    name: "x"  
  }  
  operator: "**"  
  - right: NumericLiteral {  
    + extra: {rawValue, raw}  
    value: 2  
  }  
}
```

# Трансформация

Преобразование одного синтаксического дерева в другое

$x ** 2$



$\text{Math.pow}(x, 2)$

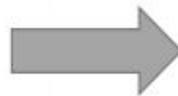
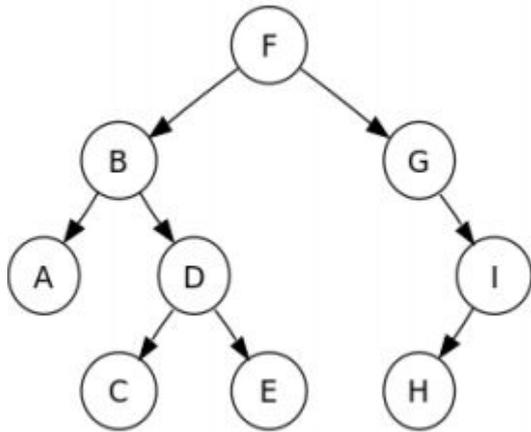
```
- left: BinaryExpression {  
  - left: Identifier = $node {  
    name: "x"  
  }  
  operator: "**"  
  - right: NumericLiteral {  
    + extra: {rawValue, raw}  
    value: 2  
  }  
}
```



```
- left: CallExpression {  
  - callee: MemberExpression {  
    - object: Identifier {  
      name: "Math"  
    }  
    - property: Identifier {  
      name: "pow"  
    }  
    computed: false  
  }  
  - arguments: [  
    - Identifier {  
      name: "x"  
    }  
    - NumericLiteral {  
      + extra: {rawValue, raw}  
      value: 2  
    }  
  ]  
}
```

# Генерация

Перевод готовой AST обратно в текст



```
"use strict";
```

```
function length(_ref) {  
  var x = _ref.x,  
      y = _ref.y;  
  
  return Math.sqrt(Math.pow(x, 2) +  
                   Math.pow(y, 2));  
}
```

# TypeScript

**TypeScript** — язык программирования, позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript.

TypeScript отличается от JavaScript:

- возможностью явного статического назначения типов;
- поддержкой использования полноценных классов (как в традиционных объектно-ориентированных языках);
- поддержкой подключения модулей.

TypeScript это расширение языка ECMAScript 5.

# TypeScript

TypeScript предлагает новый синтаксис для написания JS-приложений. Он использует аннотации типов, чтобы избавиться от некоторых проблем с типизацией в JavaScript.

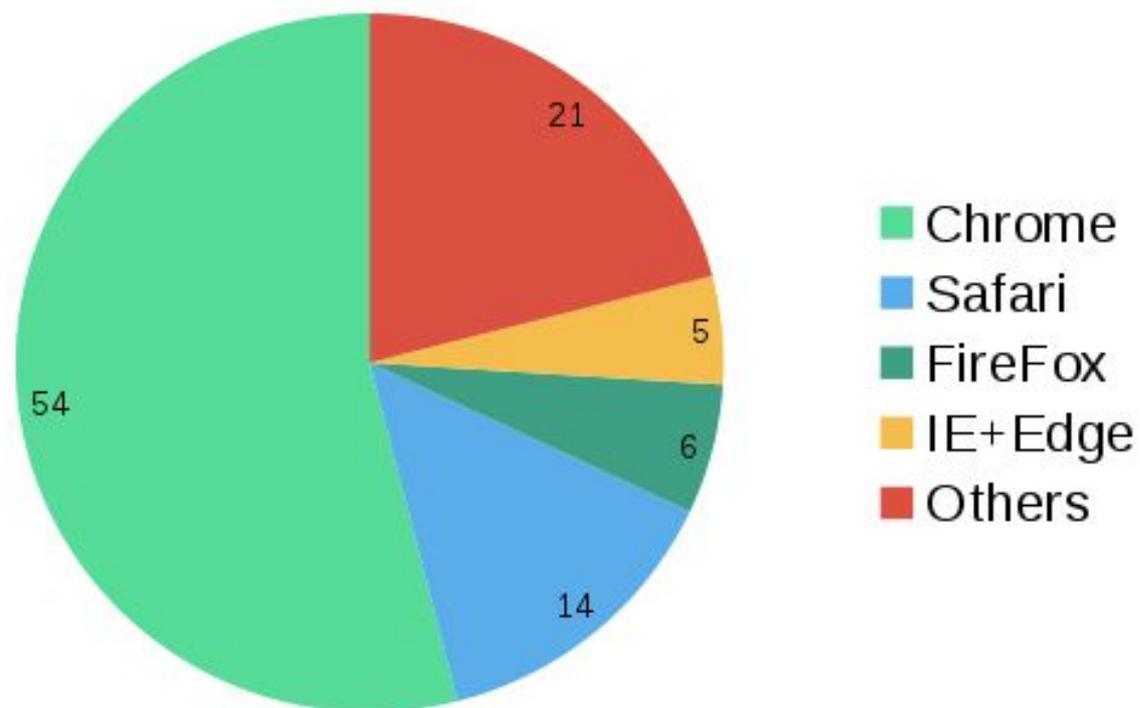
```
function greeter(person: string) {  
    return "Hello, " + person;  
}  
  
var user = "Jane User";  
  
document.body.innerHTML = greeter(user);
```

TypeScript

```
function greeter(person) {  
    return "Hello, " + person;  
}  
  
var user = "Jane User";  
  
document.body.innerHTML = greeter(user);
```

ES5

# Почему это не понадобится в ближайшее время?

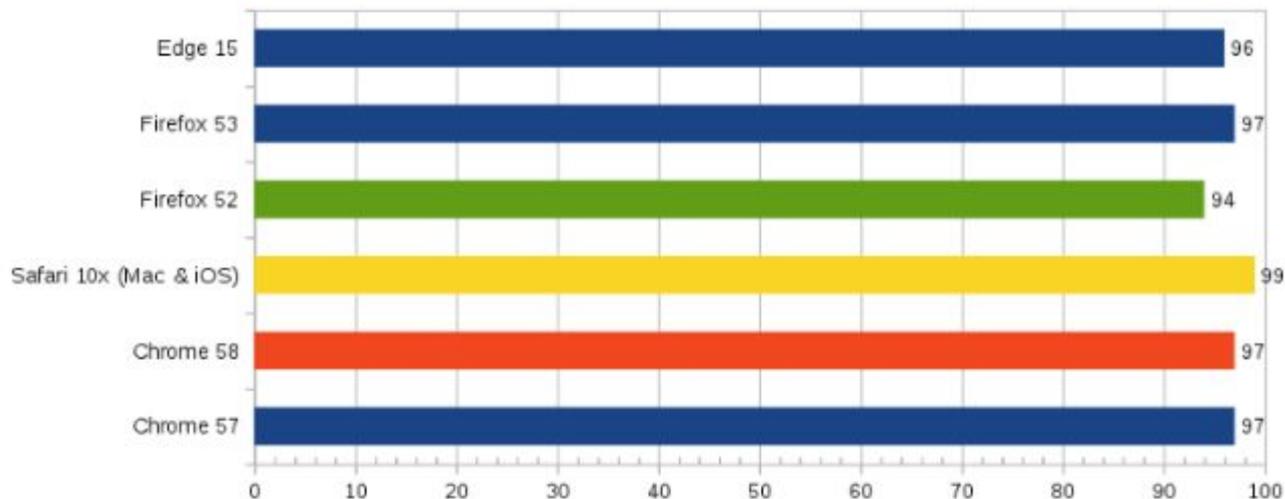


Доля рынка браузеров по состоянию на май 2017 года в%

# Почему это не понадобится в ближайшее время?

Поддержка классов появилась в браузере Chrome в 2014 году. Это позволяет браузеру работать с объявлениями классов без применения транспиляции или каких-либо вспомогательных библиотек.

На самом деле, поддержка этих механизмов браузерами — не более чем «синтаксический сахар». Эти конструкции преобразуются в те же базовые структуры, которые уже поддерживаются языком. В результате, даже если пользоваться новым синтаксисом, на более низком уровне всё будет выглядеть как создание конструкторов и манипуляции с прототипами объектов.



Все основные браузеры имеют очень хорошую поддержку ES6

# Почему это не понадобится в ближайшее время?

ES6 compatibility table

Sort by Engine types Show obsolete platforms Show unstable platforms

V8 SpiderMonkey JavaScriptCore Chakra Carakan KJS Other

Minor difference (1 point) Small feature (2 points) Medium feature (4 points) Large feature (8 points)

Feature name	Current browser	Compilers/polyfills											Desktop browsers																	
		Traceur	Babel 6	Babel 7	Babel 7	Closure	Type-Script	Type-Script	es-shim	Kong	IE 11	Edge	Edge	Edge	FF 60	FF 65	FF 66	FF 67	FF 68	CH 72	CH 73	CH 74	CH 75	SF 11.1	SF 12	SF 12.1	SF TP			
<b>Optimisation</b>																														
proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	
<b>Syntax</b>																														
default function parameters	7/7	4/7	4/7	4/7	4/7	5/7	5/7	5/7	0/7	0/7	0/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7
rest parameters	5/5	4/5	3/5	3/5	3/5	2/5	4/5	4/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
spread syntax for iterable objects	15/15	15/15	13/15	13/15	13/15	11/15	14/15	14/15	0/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	
object literal extensions	6/6	6/6	6/6	6/6	6/6	5/6	6/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
for of loops	9/9	9/9	9/9	9/9	9/9	6/9	9/9	9/9	0/9	0/9	0/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9
octal and binary literals	4/4	2/4	4/4	4/4	4/4	2/4	4/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
template literals	6/6	5/6	5/6	5/6	5/6	4/6	4/6	4/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
RegExp "y" and "u" flags	6/6	4/6	4/6	4/6	4/6	0/6	0/6	0/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
destructuring declarations	22/22	20/22	21/22	21/22	21/22	20/22	21/22	21/22	0/22	0/22	0/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22
destructuring assignment	24/24	23/24	24/24	24/24	24/24	22/24	24/24	24/24	0/24	0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24
destructuring parameters	24/24	19/24	21/24	21/24	21/24	20/24	21/24	21/24	0/24	0/24	0/24	23/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24
Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
<b>Bindings</b>																														
const	18/18	16/18	16/18	16/18	16/18	16/18	16/18	16/18	0/18	2/18	14/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	
let	14/14	12/14	12/14	12/14	12/14	12/14	12/14	12/14	0/14	0/14	12/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14
block-level function declaration	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

<https://kangax.github.io/compat-table/es6/>