EcmaScript 7 – 10

Инфиксный оператор **

```
1. var a = 10;
2. var b = 2;
3. console.log(a ** b);//100
4. console.log(Math.pow(a, b));//100

5. var a = 10.5;
6. var b = 2.5;
7. console.log(a ** b);//357.2508309997333
8. console.log(Math.pow(a, b));//357.2508309997333

9. var a = 'test';
10. var b = 2;
11. console.log(a ** b);//NaN
```

Метод includes для Array

Определяет, содержит ли массив определенный элемент, возвращая в зависимости от этого true/false

arr.includes(searchElement[, fromIndex])

```
1. console.log([1, 2, 3].includes(2));//true
2. console.log([1, 2, 3].includes(4));//false
3. console.log([1, 2, 3].includes(3, 3));//false
4. console.log([1, 2, NaN].includes(NaN));//true
5. var arr = ['a', 'b', 'c'];
6. console.log(arr.includes('c', 1));//true
7. console.log(arr.includes('c', 3));//false
8. console.log(arr.includes('c', 100));//false
```

Пример

//При отрицательных значениях поиск производится начиная
//индекса array.length - fromIndex по возрастанию

1. //длина массива равна 3
2. //fromIndex равен -1
3. //вычисленный индекс равен 3 + (-2) = 1

4. var arr = ['a', 'b', 'c'];
5. console.log(arr.includes('a', -2));//false
6. console.log(arr.includes('b', -2));//true
7. console.log(arr.includes('c', -2));//true

1. //длина массива равна 3
2. //fromIndex равен -100
3. //вычисленный индекс равен 3 + (-100) = -97

4. var arr = ['a', 'b', 'c'];
5. console.log(arr.includes('a', -100));//true
6. console.log(arr.includes('b', -100));//true
7. console.log(arr.includes('c', -100));//true

**ECMAScript 8

Паддинги строк

Метод padStart

Заполняет текущую строку другой сторокой так, что итоговая строка достигает заданной длины. Заполнение осуществляется слева текущей строки .

1. str.padStart(targetLength [, padString])
2. //targetLength - длина итоговой строки после дополнения текущей.
3. //Если значение меньше, чем длина текущей строки,
4. //текущая строка будет возвращена без изменений.
5. //padString - строка для заполнения текущей строки.
6. //Если эта строка слишком длинная для заданной длины,
7. //она будет обрезана. Значение по умолчанию - " "

Пример

1. console.log('abc'.padStart(10));//"   abc"
2. console.log('abc'.padStart(10, "foo"));//"foofoofabc"
3. console.log('abc'.padStart(6,"123465"));//"123abc"
4. console.log('abc'.padStart(8, "0"));//"00000abc"
5. console.log('abc'.padStart(1));//"abc"
6. console.log('abc'.padStart(-1));//"abc"

Метод padEnd

Заполняет текущую строку с помощью заданной строки, так чтобы результирующая строка достигла заданной длины. Дополнение применяется справа текущей строки

```
1. str.padEnd(targetLength [, padString])
2. //targetLength - длина результирующей строки,
3. //после того как текущая строка была дополнена.
4. //Если параметр меньше длины текущей строки,
5. //то будет возвращена текущая строка.
6. //padString - строка для дополнения текущей строки.
7. //Если строка слишком длинная,
8. //она будет урезана и будет применяться ее левая часть.
9. //" " - значение по умолчанию.
```

Пример

1. console.log('abc'.padEnd(10));//"abc   "
2. console.log('abc'.padEnd(10, "foo"));//"abcfoofoof"
3. console.log('abc'.padEnd(6,"123465"));//"abc123"
4. console.log('abc'.padEnd(8, "0"));//"abc00000"
5. console.log('abc'.padEnd(1));//"abc"
6. console.log('abc'.padEnd(-1));//"abc"

Асинхронные функции

Метод Object.values

```
1. //Возвращает массив значений перечисляемых свойств объекта в //том же порядке что и цикл for...in
2. // Object.values(obj) - синтаксис
3. let obj = { foo: "bar", baz: 42 };
4. console.log(Object.values(obj));//["bar", 42]
5. let obj1 = { 100: 3, 10: 2, 0: 1 };
6. console.log(Object.values(obj1));//[1, 2, 3]
7. let obj2 = { 100: "a", test: "b", 2: 10 };
8. console.log(Object.values(obj2));//[10, "a", "b"]
9. let obj3 = {test: 1, obj : {name : 'test'}};
10. console.log(Object.values(obj3));//[1, {name : "test"}]
11. console.log(Object.values("foo"));//["f", "o", "o"]
12. console.log(Object.values(1000));//[]
13. console.log(Object.values([1, 2, 3]));//[1, 2, 3]
```

Метод Object.entries

1. // Возвращает массив собственных перечисляемых свойств
   //указанного объекта в формате [key, value], в том же порядке, //что и в цикле for...in
2. // Object.entries(obj) – синтаксис
3. let obj = { foo: "bar", baz: 42 };
4. console.log(Object.entries(obj));
5. //[["foo", "bar"], ["baz", 42]]
6. let obj1 = { 100: 3, 10: 2, 0: 1 };
7. console.log(Object.entries(obj1));
8. //[["0", 1], ["10", 2], ["100", 3]]
9. let obj2 = { 100: "a", test: "b", 2: 10 };
10. console.log(Object.entries(obj2));
11. //[["2", 10], ["100", "a"], ["test", "b"]]

Еще пример

```
let obj3 = {test: 1, obj : {name : 'test'}};
console.log(Object.entries(obj3));
//[["test", 1], ["obj", [{name : "test"}]]
console.log(Object.entries("foo"));
//[["0", "f"], ["1", "o"], ["2", "o"]]
console.log(Object.entries(1000));//[]
console.log(Object.entries([1, 2, 3]));
//[["0", 1], ["1", 2], ["2", 3]]
```

'Висячие' запятые в параметрах функций

```
1.  function test1(var1, var2, var3, ) {
2.      console.log(var1);//1
3.      console.log(var2);//2
4.      console.log(var3);//3
5.      console.log(arguments);//[1, 2, 3]
6.  }
7.  test1(1, 2, 3);
8.  function test(var1, , ) {
9.      console.log(var1);
10.     console.log(arguments);
11. }
12. test(1, 2, 3);
13. //Ex. Uncaught SyntaxError: Unexpected token ,
```

'Висячие' запятые при вызове функций

```
1.  function test1(var1, var2, var3) {
2.      console.log(var1);//1
3.      console.log(var2);//2
4.      console.log(var3);//3
5.      console.log(arguments);//[1, 2, 3]
6.  }
7.  test1(1, 2, 3, );
8.  function test(var1) {
9.      console.log(var1);
10.     console.log(arguments);
11. }
12. test(1, , , );
13. //Ex. Uncaught SyntaxError: Unexpected token ,
```

'Висячие' запятые и...

```
let obj = { a:'b', b: 'c', }
console.log(obj);//{a: "b", b: "c"}
const obj1 = {
    print: function (){console.log(this);}
    ,
}
obj1.print();//{print: ƒ}
var arr =[1, 2, 3, ];//length = 3
console.log(arr);//[1, 2, 3]
console.log(arr[3]);//undefined
var arr1 = [1, 2, 3, , , ];//length = 5
console.log(arr1);//[1, 2, 3, empty × 2]
console.log(arr1[4]);//undefined
```

Метод Object.getOwnPropertyDescriptors

- Возвращает дескриптор свойства переданного объекта. Свойство должно быть определено непосредственно в объекте, а не получено через цепочку прототипов
- Синтаксис:
- Object.getOwnPropertyDescriptor(obj, prop)

Пример

- const obj = {
- name: 'Test',
- get getName() {
- return this.name;
- }
- };
- console.log(Object.getOwnPropertyDescriptor(obj, 'getName'));
- //{
- // configurable: true,
- // enumerable: true,

ES9

Spread и rest операторы с объектами

- rest и spread появились в ES6 и могут быть использованы для работы с массивами. Однако теперь можно работать и с объектами
- const obj1 = { one, two, ...others }

Пример

- const { one, two, ...others } =
- { one: 1, two: 2, three: 3, four: 4, five: 5 }
- console.log(one) *//1*
- console.log(two) *//2*
- console.log(others) *//{ three: 3, four: 4, fife: 5 }*

Асинхронные итераторы

- Конструкция for-await-of позволяет вызывать асинхронные функции, возвращающие промисы, в циклах. Такие циклы ожидают разрешения промиса перед переходом к следующему шагу.
- for await (const line of readLines(filePath))
- { console.log(line) }

Метод Promise.prototype.finally()

- Если промис успешно разрешается — вызывается then(). Если найдена ошибка — вызывается метод catch().
- Метод finally() позволяет выполнять некий код независимо от того, что сработало до этого.

Метод Promise.prototype.finally()

- fetch('/common/getfile.json')
- .then(data => data.json())
- .catch(error => console.error(error))
- .finally(() => console.log('End work programm!'))

Некоторые улучшения регулярных выражений

- Возможность опережающих проверок, использующая конструкцию ?= вместо ?<=

Пример 1

- const r = /Roger(?= Waters)/
- const res1 = r.test('Roger is my dog')
- const res2 = r.test('Roger is my dog and Roger Waters is a famous musician')
- console.log(res1) *//false*
- console.log(res2) *//true*

Пример 2

- const r = /Roger(?! Waters)/g
- const res1 = r.test('Roger is my dog')
- const res2 = r.test('Roger is my dog and Roger Waters is a famous musician')
- console.log(res1) *//true*
- console.log(res2) *//false*

Флаг регулярных выражений s

- Использование флага s приводит к тому, что символ . (точка) будет, кроме прочих, соответствовать и символу новой строки.

- console.log(/hello.world/.test('hello\nworld')) *// false*
  console.log(/hello.world/s.test('hello\nworld')) *// true*

Object.fromEntries()

- Метод **Object.fromEntries()** преобразует список пар ключ-значение в объект.

Пример

- const entries = new Map([
- ['foo', 'bar'],
- ['baz', 42]
-]);

- const obj = Object.fromEntries(entries);

- console.log(obj);
- // Object { foo: "bar", baz: 42 }

matchAll()

- Метод matchAll() возвращает итератор по всем результатам при сопоставлении *строки с регулярным выражением*.

Пример

- let regexp = /t(e)(st(\d?))/g;
- let str = 'test1test2';

- let array = [...str.matchAll(regexp)];

- console.log(array[0]);
- // expected output: Array ["test1", "e", "st1", "1"]

- console.log(array[1]);
- // expected output: Array ["test2", "e", "st2", "2"]

trimStart() и trimEnd()

- trimStart() удаляет пробелы из начала строки. trimLeft() является псевдонимом этого метода.
- В trimEnd() удаляет пробелы из конца строки. trimRight() является псевдонимом этого метода.
- trim() удаляет пробельные символы с обоих концов строки.

Пример

- var str = '   foo ';

  console.log(str.length); // 8

  str = str.trimStart();
  console.log(str.length); // 5
  console.log(str);   // 'foo '

Symbol.Description

- Свойство только для чтения description- это строка, возвращающая необязательное описание Symbol объектов.

Пример

- Symbol('desc').toString(); // "Symbol(desc)"
- Symbol('desc').description; // "desc"
- Symbol('').description; // ""
- Symbol().description; // undefined
-
  // global symbols

- Symbol.for('foo').toString(); // "Symbol(foo)"

- Symbol.for('foo').description; // "foo"

JSON ⊂ ECMAScript

- Символы неэкранированного разделителя строк U+2028 и разделителя абзацев U+2029 не были представлены в предыдущих версиях ECMAScript.
- U+2028 — разделитель абзацев.
- U+2029 — разделитель строк.

Устойчивый метод sort()

- Метод sort() *на месте* сортирует элементы массива и возвращает отсортированный массив.

Пример

- let scores = [1, 2, 2, 10, 21];
- scores.sort(); // [1, 10, 2, 2, 21]

- var fruit = ['арбузы', 'бананы', 'Вишня'];
- fruit.sort(); // ['Вишня', 'арбузы', 'бананы']

BigInt

- BigInt создается путем добавления n в конец целочисленного литерала — 10n — или вызовом функции BigInt().
- В некотором смысле он похож на Number, но отличается в некоторых ключевых моментах — его нельзя использовать с методами во встроенном объекте Math и нельзя смешивать в операциях с любыми экземплярами Number.

BigInt

- Number и BigInt нельзя смешивать в операциях — они должны быть приведены к тому же типу.
- Будте осторожны приводя значения туда и обратно, так как точность BigInt может быть потеряна при приведении к числу (Number).

BigInt

- const theBiggestInt = 9007199254740991n;

- const alsoHuge = BigInt(9007199254740991);
  // ↪ 9007199254740991n

- const hugeString = BigInt("9007199254740991");
  // ↪ 9007199254740991n

- const hugeHex = BigInt("0x1fffffffffffff");
  // ↪ 9007199254740991n

- const hugeBin = BigInt("0b11111111111111111111111111111111111111111111111111111");
  // ↪ 9007199254740991n

BigInt

- typeof 1n === 'bigint'; // true
- typeof BigInt('1') === 'bigint'; // true

- typeof Object(1n) === 'object'; // true

BigInt

- Следующие операторы могут использоваться с BigInts (или объектом-оберткой BigIntc): +, *, -, **, %.
- const previousMaxSafe = BigInt(Number.MAX_SAFE_INTEGER);
  // ↪ 9007199254740991
- const maxPlusOne = previousMaxSafe + 1n;
  // ↪ 9007199254740992n
- const subtr = multi – 10n;
  // ↪ 1801439850948197 2n
- const mod = multi % 10n;
  // ↪ 2n
- const bigN = 2n ** 54n;
  // ↪ 18014398509481984n
- bigN * -1n

BigInt

- 0n === 0 // ↪ false
- 0n == 0 // ↪ true
- 1n < 2 // ↪ true
- 2n > 1 // ↪ true
- 2 > 2 // ↪ false
- 2n > 2 // ↪ false

BigInt

- const mixed = [4n, 6, -12n, 10, 4, 0, 0n];
- // ↪ [4n, 6, -12n, 10, 4, 0, 0n]

- mixed.sort();
- // ↪ [-12n, 0, 0n, 10, 4n, 4, 6]

BigInt

- if (0n) {
  console.log('Hello from the if!');
  } else {
  console.log('Hello from the else!');
  }// ↪ "Hello from the else!"
- 0n || 12n // ↪ 12n
- 0n && 12n // ↪ 0n
- Boolean(0n) // ↪ false
- Boolean(12n) // ↪ true
- !12n // ↪ false
- !0n // ↪ true

BigInt

- BigInt.asIntN()
- Оборачивает BigInt в пределах от -2width-1 до 2width-1-1

- BigInt.asUintN()
- Оборачивает a BigInt в пределах от 0 до 2width-1

- BigInt.prototype
- Позволяет добавлять свойства к объекту BigInt.

Динамический импорт

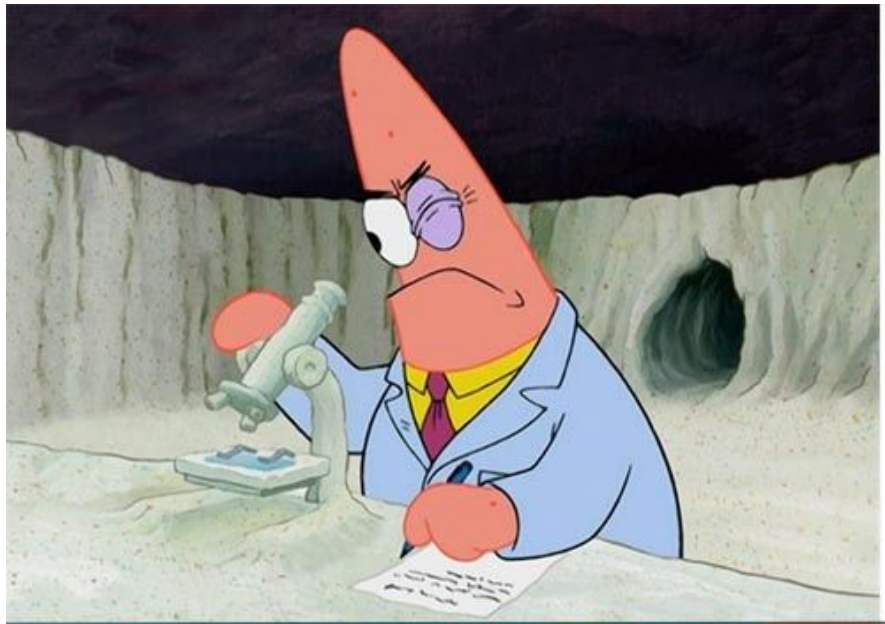
- Динамический import() возвращает промис для объекта пространства имен запрашиваемого модуля. Следовательно, теперь импорт можно назначить переменной, используя async/await.

Динамический импорт

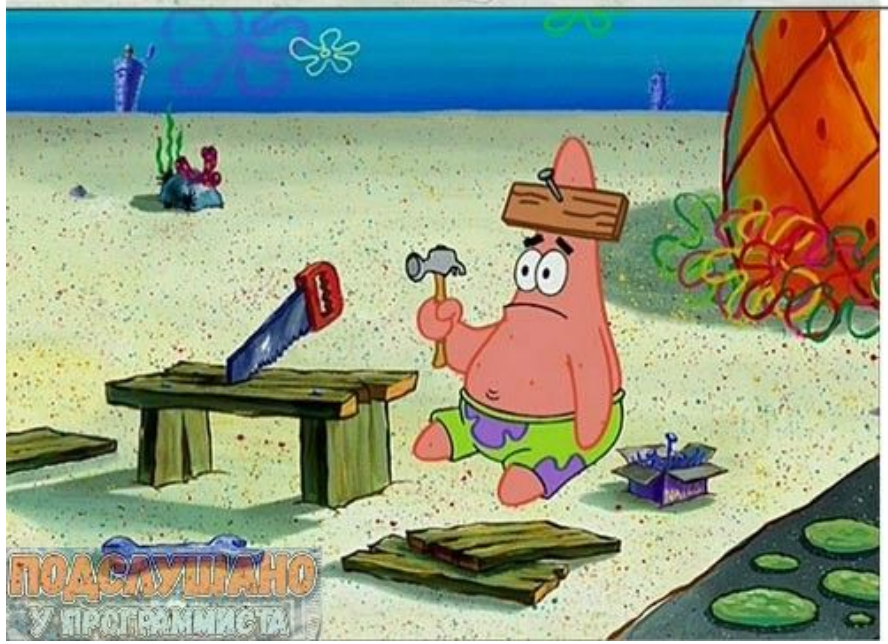
- //one.js
- export function hi() {
- console.log(`Привет`);
- }
- export function bye() {
- console.log(`Пока`);
- }
- export default function() {
- console.log("Модуль загружен (экспорт по умолчанию)!");
- }

Динамический импорт

- main.js
- async function load() {
  let say = await import('./say.js');
  say.hi(); // Привет!
  say.bye(); // Пока!
  say.default(); // Модуль загружен (экспорт по умолчанию)!
  }



Программирование в теории



Программирование на практике