Знакомство с ES6

ES6

- let&const
- Деструктуризация (массивов и объектов)
- …Spread-оператор
- Стрелочные функции
- Шаблонные строки

Метод includes

Проверяет, включает ли одна строка str в себя другую строку searchString, возвращает true/false

```
1. let str = 'To be, or not to be, that is the question.';

2. console.log(str.includes('To be'));       //true

3. console.log(str.includes('question'));    //true

4. console.log(str.includes('nonexistent')); //false

5. console.log(str.includes('To be', 1));    //false

6. console.log(str.includes('TO BE'));       //false
```

Метод endsWith

Возвращает true, если строка str заканчивается подстрокой searchString

```
1. let str = 'To be, or not to be, that is the question.';

2. console.log(str.endsWith('question.'));//true

3. console.log(str.endsWith('to be'));    //false

4. console.log(str.endsWith('to be', 19));//true
```

Метод startsWith

Возвращает true, если строка str начинается со строки searchString

```
let str = 'To be, or not to be, that is the question.';

console.log(str.startsWith('To be'));       //true

console.log(str.startsWith('not to be'));    //false

console.log(str.startsWith('not to be', 10));//true
```

Метод repeat

Повторяет строку str count раз

```
'abc'.repeat(-1);   //RangeError

'abc'.repeat(0);    //''

'abc'.repeat(1);    //'abc'

'abc'.repeat(2);    //'abcabc'

'abc'.repeat(3.5);  //'abcabcabc' (count will be converted to integer)

'abc'.repeat(1/0);  //RangeError
```

Объекты и прототипы

Короткое свойство

При объявлении свойства объекта достаточно указать только его имя, а значение будет взято из переменной с аналогичным именем

```
1. let name = "Вася";
2. let isAdmin = true;

3. let user = {
4.   name,
5.   isAdmin
6. };

7. console.log(JSON.stringify(user));//{"name": "Вася", "isAdmin": true}
```

Вычисляемые свойства

```
1. let propName = "firstName";

2. let user = {

3.     [propName]: "Вася"

4. };

5. console.log(user.firstName);//Вася
```

Вычисляемые свойства

```
1.  let a = "Зелёный ";

2.  let b = "Крокодил";

3.  let user = {

4.      [(a + b).toLowerCase()]: "Петя"

5.  };

6.  console.log( user["зелёный крокодил"] );//Петя
```

Метод setPrototypeOf

Метод устанавливает прототип (внутреннее свойство [[Prototype]]) указанного объекта в другой объект или null

```
let dict = Object.setPrototypeOf({}, null);

let person = {
    name: 'unknown'
};
let student = {
    group: 1
};
let p1 = Object.setPrototypeOf(student, person);
console.log(p1.group);//1
console.log(p1.name);//unknown
```

Object.assign

Метод используется для копирования значений всех собственных перечисляемых свойств из одного или более объектов в целевой объект

```
1. Object.assign(target, src1, src2...)
2. let o1 = { a: 1 };
3. let o2 = { b: 2 };
4. let o3 = { c: 3 };

5. let obj = Object.assign(o1, o2, o3);

6. console.log(obj);//{ a: 1, b: 2, c: 3 }

7. console.log(o1); //{ a: 1, b: 2, c: 3 }

8. //изменился и целевой объект
```

Object.is

Метод определяет, являются ли два значения одинаковыми

```
1.  Iet sSame = Object.is(value1, value2);
2.  Object.is('foo', 'foo');   //true
3.  Object.is(window, window); //true
4.  Object.is('foo', 'bar');   //false
5.  Object.is([], []);        //false

6.  let test = { a: 1 };
7.  Object.is(test, test);     //true
8.  Object.is(null, null);     //true
9.  //Специальные случаи

10. Object.is(0, -0);         //false
11. Object.is(-0, -0);        //true
12. Object.is(NaN, 0/0);      //true
13. Object.is(NaN, NaN);      //true
```

Объявление метода

Более короткий и удобный синтаксис

```
let name = "Вася";
let user = {
    name,
    //something: function()
    //{
    //console.log(this.name);
    //}
    something() {
        console.log(this.name);
    }
};

user.something();//Вася
```

Классы

Объявление класса

```
1.  class Название [extends Родитель]  {
2.      constructor
3.      методы
4.  }
```

Объявление класса

```
1.  class Polygon {
2.      constructor(height, width) {
3.          this.height = height;
4.          this.width = width;
5.      }
6.  }
```

Выражение класса

```
1.  //безымянный
2.  var Polygon = class {
3.      constructor(height, width) {
4.          this.height = height;
5.          this.width = width;
6.      }
7.  };
8.  //именованный
9.  var Polygon = class Polygon {
10.     constructor(height, width) {
11.         this.height = height;
12.         this.width = width;
13.     }
14. };
```

Создание объекта и прототип

Constructor запускается при создании new Object, остальные методы записываются в Object.prototype

```
1. class User {
2.     constructor(name) {
3.         this.name = name;
4.     }
5.     sayHi() { console.log(this.name);}
6. }
7. let user = new User("Вася");
```

Создание объекта и прототип

```
1. function User(name) {
2.     this.name = name;
3. }
4. User.prototype.sayHi = function() {
5.     console.log(this.name);
6. };
```

Всплытие (hoisting)

Разница между объявлением функции (function declaration) и объявлением класса (class declaration) в том, что объявление функции совершает подъём (hoisted), в то время как объявление класса — нет

```
1. var p = new Polygon();

2. class Polygon {}

3. //Uncaught ReferenceError: Polygon is not defined
```

Статические методы

```
1.  class Point {
2.      constructor(x, y) {
3.          this.x = x;
4.          this.y = y;
5.      }
6.      static distance(a, b) {
7.          const dx = a.x - b.x;
8.          const dy = a.y - b.y;
9.          return Math.sqrt(dx*dx + dy*dy);
10.     }
11. }
12. const p1 = new Point(5, 5);
13. const p2 = new Point(10, 10);
14. console.log(Point.distance(p1, p2));//7.07....
```

Геттеры, сеттеры

```
class User {
    constructor(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    get fullName() {
        return `${this.firstName} ${this.lastName}`;
    }
    set fullName(newValue) {
        [this.firstName, this.lastName] = newValue.split(' ');
    }
};
let user = new User('Maksim', 'Hladki');
console.log(user.fullName);//Maksim Hladki
user.fullName = "Ivan Ivanov";
console.log(user.fullName);//Ivan Ivanov
```

Пример

```
1.  class Rectangle {
2.      constructor (width, height) {
3.          this._width  = width
4.          this._height = height
5.      }
6.      set width  (width)  { this._width = width}
7.      get width  ()       { return this._width}
8.      set height (height) { this._height = height}
9.      get height ()       { return this._height}
10.     get area   ()       { return this._width * this._height }
11. }
12. let test = new Rectangle(50, 20);
13. console.log(test.area);//1000
```

Вычисляемые имена методов

```
class Foo() {
    myMethod() {}
}
```

```
class Foo() {
    ['my'+'Method']() {}
}
```

```
const m = 'myMethod';
class Foo() {
    [m]() {}
}
```

Наследование

```
1.     //Только один конструктор, прототип, базовый класс!
2.
3.    class Child extends Parent {
4.
5.    //TODO logic
6.
7.    }
```

Пример

```
class Point {
    constructor(x, y) {
        this.x = x; this.y = y;
    }
    toString() {
        return '(' + this.x + ', ' + this.y + ')';
    }
}
class ColorPoint extends Point {
    constructor(color) {
        super(0, 0);
        this.color = color;
    }
    toString() {
        return super.toString() + ' in ' + this.color;
    }
}
let cPoint = new ColorPoint('red');
console.log(cPoint.toString());//(0, 0) in red
```

Наследование статических методов

```
1.  class Foo {
2.      static classMethod() {
3.          return 'hello';
4.      }
5.  }
6.  class Bar extends Foo {
7.      //TODO
8.  }
9.
0.  console.log(Bar.classMethod());//hello
```

Super

```
1. //Используется для вызова функций, принадлежащих
2. //родителю объекта
3.
4. super([arguments]);//вызов родительского конструктора
5.
6. super.functionOnParent([arguments]);
```

Пример: вызов конструктора

```
class Polygon {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }
}
class Square extends Polygon {
    constructor(length) {
        super(length, length);
    }
    get area() {
        return this.height * this.width;
    }
}
```

Вызов метода

```
1.  class Foo {
2.      static classMethod() {
3.          return 'hello';
4.      }
5.  }
6.  class Bar extends Foo {
7.      static classMethod() {
8.          return super.classMethod() + ', too';
9.      }
0.  }
1.  Bar.classMethod();//hello, too
```

Mixins

Mixins

```
1.  // Абстрактные подклассы (mix-ins) — это шаблоны для классов.
2.  //У класса может быть только один родительский класс, поэтому
3.  // множественное наследование невозможно.
4.  //Функциональность должен предоставлять родительский класс
5.
6.  class B extends A, M {}//      Uncaught SyntaxError: Unexpected token ,
7.                          //      множественного наследования нет

8.  const mixin = base => class extends base {
9.      /* свойства и методы */
0.  }
```

Пример

```
1.  class Person { ··· }
2.  const Storage = Sup => class extends Sup {
3.      save(database) { ··· }
4.  };
5.  const Validation = Sup => class extends Sup {
6.      validate(schema) { ··· }
7.  };
8.  class Employee extends Storage(Validation(Person)) { ··· }
```

Пример

```
let MyMixin = (superclass) => class extends superclass {
    test() {
        console.log('test from MyMixin');
    }
};

class MyClass extends MyMixin(MyBaseClass) {
    /* ... */
}
let c = new MyClass();
c.test();//test from MyMixin
```

Symbol

Тип данных Symbol

Уникальный и неизменяемый тип данных, который может быть использован как идентификатор для свойств объектов

Символьный объект — это объект-обертка для примитивного символьного типа

```
let sym = Symbol("foo");

console.log(typeof sym);//symbol

let symObj= Object(sym);

console.log(typeof symObj);//object

console.log(Symbol("name") == Symbol("name"));//false
```

Тип данных Symbol

```
let isAdmin = Symbol("isAdmin");

let user = {
  name: "Вася",
  [isAdmin]: true
};

console.log(user[isAdmin]);//true
```

Тип данных Symbol

Свойство, объявленное через символ, не будет видно в for-in, Object.keys, Object.getOwnPropertyNames, также не будет добавлено при использовании JSON.stringify

Тип данных Symbol

```
let user = {
  name: "Вася",
  age: 30,
  [Symbol.for("isAdmin")]: true
};

//в цикле for..in не будет символа

console.log(Object.keys(user));//["name", "age"]

//доступ к свойству через глобальный символ — работает

console.log(user[Symbol.for("isAdmin")]);//true
```

Глобальные символы

Глобальный реестр символов позволяет иметь общие глобальные символы, которые можно получить из реестра по имени. Используется метод for

```
//создание символа в реестре

let name = Symbol.for("name");

//символ уже есть, чтение из реестра

console.log(Symbol.for("name") == name);//true
```

Встроенные символы (Well-known)

Symbol.iterator	возвращающий итератор для объекта
Symbol.match	сопоставление объекта со строкой (String.prototype.match)
Symbol.replace	заменяет совпавшие подстроки в строке (String.prototype.replace)
Symbol.search	возвращает индекс вхождения подстроки, соответствующей регулярному выражению (String.prototype.search)
Symbol.split	разбивает строку на части в местах, соответствующих регулярному выражению (String.prototype.split)
Symbol.for(key)	ищет существующие символы по заданному ключу и возвращает его, если он найден
Symbol.species	определяет конструктор для порожденных объектов