

Знакомство с ES6

ES6

- let&const
- Деструктуризация (массивов и объектов)
- ...Spread-оператор

Функции

Параметры по умолчанию

```
1 function [name]([param1[= defaultValue1][, ..., paramN[= dValueN]])
2 {
3     //function's body
4 }
```

```
1 function multiply(a, b = 1) {
2     return a*b;
3 }
4 console.log(multiply(5)); //5
```

```
1 function multiply(a, b = 1) {  
2     return a*b;  
3 }  
4 multiply(5, undefined); //5
```

```
1 function append(value, array = []) {  
2     array.push(value);  
3     return array;  
4 }  
5 append(1); // [1]  
6 append(2); // [2]
```

```
1 function getRandomInt(min, max) {  
2   |   return Math.floor(Math.random() * (max - min)) + min;  
3   | }  
4 function print(number = getRandomInt(10, 100)){  
5   |   console.log(number);  
6   | }  
7 print();//29
```

Оператор spread

В нашем случае, rest - это массив, а значит, можно использовать методы map, forEach и т.д.

```
1 function printName(firstName, lastName, ...rest) {  
2     console.log(firstName + ' ' + lastName + ' - ' + rest);  
3     console.log(rest);  
4 }  
5 printName('Иван', 'Иванов', 'Иванович', '1959');  
6 //Иван Иванов - Иванович,1959  
7 //["Иванович", "1959"]
```

Свойство name

```
1 function getRandomInt(min, max)
2 {
3     return Math.floor(Math.random() * (max - min)) + min;
4 }
5 console.log(getRandomInt.name); //getRandomInt
6 getRandomInt.name = 'f';
7 console.log(getRandomInt.name); //getRandomInt
8 console.log(f.name); //ReferenceError: f is not defined
```


Стрелочные функции

Стрелочные функции

Выражения имеют более короткий синтаксис, всегда анонимные и лексически привязанные к значению `this`.

Синтаксис: $(\text{param1}, \text{param2}, \text{paramN}) \Rightarrow \text{expression}$

Особенности использования стрелочных функций

- Лексическое связывание. Значения `this`, `super` и `arguments` определяются не тем, как стрелочные функции были вызваны, а тем, как они были созданы
- Неизменяемые `this`, `super` и `arguments`. Значения этих переменных внутри стрелочных функций остаются неизменными на протяжении всего жизненного цикла функции
- Стрелочные функции не могут быть использованы как конструктор
- Недоступность «собственного» значения переменной `arguments`, `this` ...

Короткая запись

```
1  var Specialization = [  
2      "PFE",  
3      "FE"  
4  ];  
5  
6  var a = faculties.map(function(s){ return s.length });//[3,2]  
7  var b = faculties.map( s => s.length);//[3,2]
```

Пример

```
1  let square = x => x*x;
2  console.log(square(3)); //9
3
4  let sum = (x, y) => x + y;
5  console.log(sum(3,4)); //7
6
7  let getObject = () => ({ brand: 'BMW' });
8  console.log(getObject()); //Object {brand: "BMW"}
9
10 let func = () => 77;
11 console.log(func()); //77
```

Отсутствие Arguments

```
1 function foo() {  
2     var f = (i) => arguments[0]+i;  
3     //возмет arguments из функции foo  
4     console.log(f(2));  
5 }  
6 foo(1); //3
```

Отсутствие запуска с new

```
1  var a = new (function() {});  
2  //переменной a будет присвоено значение экземпляра анонимной функции  
3  var b = new (() => {});  
4  //TypeError: (intermediate value) is not a constructor
```

Строки

Шаблонные строки

Выглядят как обычные строки, за исключением того, что обернуты символами обратных кавычек `

```
`строка текста`
```

```
`строка текста 1
```

```
строка текста 2`
```

```
`строка текста ${выражение} строка текста`
```

```
tag `строка текста ${выражение} строка текста`
```

Многострочные литералы

```
console.log(`string text line 1
```

```
string text line 2`);
```

```
/*string text line 1
```

```
// string text line 2*/
```

Интерполяция выражений

В строке создаётся конструкция $\{...\}$, внутри которой вы можете поместить любую переменную или выражение.

Строки, созданные с помощью обычных кавычек (' и ") не поддерживают интерполяцию. Для поддержки интерполяции следует использовать обратную кавычку ` (клавиша \grave{e} на клавиатуре):

С помощью интерполяции в строку можно поместить результат выполнения любого выражения, например, вызов функции

Интерполяция выражений

1. `let arg1 = 2;`

2. `let arg2 = 3;`

3. `console.log(`${arg1} + ${arg2} = ${arg1 + arg2}`); // 2 + 3 = 5`

Интерполяция выражений

*Скорее всего, будут возникать ситуации, когда одного уровня интерполяции будет недостаточно. В подобных случаях удобно пользоваться вложенностью (интерполяция внутри интерполяции). Следует помнить, что весь код, находящийся внутри $\{ \dots \}$ интерпретируется, как отдельное выражение, то есть может содержать обратные кавычки, которые не будут восприняты, как конец строки

Улучшенная поддержка юникода

Используется кодировка UTF-16. На хранение одного символа необходимо 2 байта

```
console.log( '我'.length );//1  
  
console.log( '鯁'.length );//2  
  
//расширенная поддержка метасимволов, математических символов  
  
//и смайликов  
  
console.log( 'ℳ'.length );//2, MATHEMATICAL SCRIPT CAPITAL X  
  
console.log( '😂'.length );//2, FACE WITH TEARS OF JOY
```

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

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`

```
1. let dict = Object.setPrototypeOf({}, null);
2.
3. let person = {
4.   name: 'unknown'
5. };
6. let student = {
7.   group: 1
8. };
9. let p1 = Object.setPrototypeOf(student, person);
0. console.log(p1.group);//1
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. let 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

0. Object.is(0, -0); //false
1. Object.is(-0, -0); //true
2. Object.is(NaN, 0/0); //true
3. Object.is(NaN, NaN); //true
```

Объявление метода

Более короткий и удобный синтаксис

```
1. let name = "Вася";
2. let user = {
3.   name,
4.   //something: function()
5.   //{
6.   //console.log(this.name);
7.   //}
8.   something() {
9.     console.log(this.name);
0.   }
1. };
2. user.something();//Вася
```