

JavaScript

LEVEL UP: _ => _

Functions: default parameters

Параметры по умолчанию – синтаксис, который позволяет задать аргументам функции значения по умолчанию, если функция была вызвана без аргумента или со значением `undefined`.

```
function functionName(arg1=defaultValue1, arg2=defaultValue2) {...}
```

Этот код будет эквивалентен следующему:

```
function functionName(arg1, arg2) {  
  arg1= arg1 || defaultValue1;  
  arg2= arg2 || defaultValue2;  
}
```

Functions: default parameters

Пример использования:

```
function printText(message = 'Hello', name = 'User') {  
  console.log(` ${message}, ${name} `);  
}
```

```
printText(); // Hello, User
```

```
let msg, name;
```

```
printText(msg, name); // Hello, User
```

```
printText('Ola', 'Amigo'); // Ola, Amigo
```

Arrow functions

Функции-стрелки (стрелочные функции, функции-ракеты) – новый синтаксис для создания функциональных выражений.

Особенности arrow function:

- 1) более лаконичный синтаксис;
- 2) отсутствие псевдомассива аргументов `arguments`;
- 3) лексическое определение `this`;
- 4) не могут использоваться в качестве конструкторов (с оператором `new`);
- 5) не могут использоваться для создания генераторов.

Arrow functions: syntax

Arrow function всегда создаётся с помощью function expression и не имеет имени (анонимная). Для создания arrow functions не используется ключевое слово `function` и вместо этого используется fat arrow (жирная стрелка) `=>`

```
const foo = () => { return 2 + 3; } →  
// function foo() { return 2 + 3; }
```

```
const boo = () => {  
  const number = Math.random(); return number;  
} →  
/* function boo() {  
  const number = Math.random(); return number;  
} */
```

Arrow functions: syntax

Если тело функции содержит однострочное выражение, то фигурные скобки можно опустить, и слово `return` в данном случае тоже опускается:

```
const test = () => 2 + 3;  
// function test() { return 2 + 3; }
```

Однако если в функции более одной строки, то следует использовать традиционный синтаксис:

```
const test = () => {  
  const pi = Math.PI;  
  return pi * 2;  
}
```

Arrow functions: syntax

Если функция принимает один аргумент, то круглые скобки вокруг аргумента можно не использовать:

```
const getModule = (number) => { return Math.abs(number); }
```

→

```
const getModule = number => { return Math.abs(number); }
```

→

```
const getModule = number => Math.abs(number);
```

Однако при отсутствии аргументов или наличии более двух аргументов, круглые скобки обязательны.

Arrow functions: syntax

<pre>function <i>getText</i>() { return 'Hello!'; }</pre>	1) const <i>getText</i> = () => { return 'Hello!'; } 2) const <i>getText</i> = () => 'Hello!';
<pre>function <i>getGreet</i>(name) { return 'Hello, ' + name; }</pre>	1) const <i>getGreet</i> = (name) => { return 'Hello, ' + name; } 2) const <i>getGreet</i> = (name) => 'Hello, ' + name; 3) const <i>getGreet</i> = name => { return 'Hello, ' + name; } 4) const <i>getGreet</i> = name => 'Hello, ' + name;

Arrow functions: syntax

Если стрелочная функция используется без фигурных скобок и слова `return`, и при этом мы хотим вернуть объект, то объект нужно заключить в круглые скобки:

```
const getTextInfo = (text) => {  
  return { length: text.length, isEven: !(text.length % 2) };  
}  
getTextInfo('ola!'); // {length: 4, isEven: true}
```

```
const getTextInfo = text => ({ length: text.length, isEven: !(text.length % 2) });  
getTextInfo('hello'); // {length: 5, isEven: false}
```

Arrow functions: syntax

Функции-стрелки очень удобно использовать в качестве колбэков, например, при переборе массива. Сравните:

```
[0,1,2,3,4,5,6,7].filter(function (number) { return number % 2});  
// [1, 3, 5, 7]
```

```
[0,1,2,3,4,5,6,7].filter(number => number % 2);  
// [1, 3, 5, 7]
```

```
[0,1,2,3].map(number => ({ digit: number}));  
// [ {digit: 0}, {digit: 1}, {digit: 2} ...]
```

Arrow functions: arguments

Внутри обычных функций можно обратиться к псевдомассиву `arguments`, который содержит список параметров, с которыми была вызвана функция:

```
function printParams() {  
  console.log(arguments);  
}  
printParams('test', 123); // ['test', 123]
```

Внутри arrow function такая возможность отсутствует:

```
const printParams = () => console.log(arguments);  
printParams('test', 123);
```

Uncaught ReferenceError: arguments is not defined

Arrow functions: arguments

Проблему отсутствия arguments в стрелочных функциях можно решить путём сочетания функций-стрелок с rest оператором:

```
const printParams = (...props) => console.log(props);  
printParams('test', 123); // ["test", 123]
```

В данном случае переменная props будет истинным массивом, так как rest оператор всегда возвращает нативный массив (не псевдомассив).

Arrow functions: this

При создании функций традиционным синтаксисом ключевое слово this определяется в момент вызова функции и зависит от способа вызова функции:

`function getThis() { console.log(this); }`

<code>getThis();</code> <i>Window</i> <i>undefined в strict</i>	<code>const user = { getThis: getThis, age: 25 };</code> <code>user.getThis();</code> <code>{age: 25, getThis:...}</code>	<code>getThis.call({name: 'Albus' });</code> <code>{name: "Albus"}</code>	<code>getThis = getThis.bind({ age: 18 });</code> <code>getThis();</code> <code>{age: 18}</code>	<code>new getThis();</code> <code><i>getThis</i> {}</code>
---	--	---	--	--

Arrow functions: this

Для стрелочных функций не существует “собственного” `this`, то есть они используют `this` “выше” уровнем – тот `this`, который определяется в момент создания актуального окружения, в котором находится стрелочная функция.

```
const user = { age: 45,  
               getAge: () => { return this.age; }  
               };  
user.getAge(); // undefined
```

В данном случае объект `user` создавался в глобальном пространстве, следовательно, в момент создания объекта `this` являлся глобальным объектом (`Window`).

Arrow functions: this

Сравним вызов двух функций:

```
const tools = {  
  value: 45,  
  getValue: function () {  
    console.log(this);  
    return this.value;  
  }  
};
```

tools.getValue(); →
{age: 45, getValue: function}
45

Контекст определён в момент вызова

```
const utils = {  
  value: 45,  
  getValue: () => {  
    console.log(this);  
    return this.value;  
  }  
};
```

utils.getValue(); →
Window
undefined

Контекст определён во время создания функции

Arrow functions: this

```
function Menu(title = 'Main') {  
  this.title = title;  
  this.getTitle = function () { return this.title; }  
}
```

```
const homeMenu = new Menu('home');  
homeMenu.getTitle(); // "home"
```

```
const getTitle = homeMenu.getTitle;  
getTitle(); // undefined
```

В последнем случае произошла потеря контекста, так как функция *function () { return this.title; }* была вызвана в общем контексте (в глобальном пространстве), и *this* для неё - объект *Window*

```
function Menu(title = 'Main') {  
  this.title = title;  
  this.getTitle = () => this.title;  
}
```

```
const homeMenu = new Menu('home');  
homeMenu.getTitle(); // "home"
```

```
const getTitle = homeMenu.getTitle;  
getTitle(); // "home"
```

В данном примере стрелочная функция создавалась в момент вызова функции *Menu*, соответственно, *this* для неё - это *this* внутри функции *Menu* (то есть экземпляр класса *Menu*). Поэтому в последнем примере контекст не потерян.

Arrow functions: this

Вспомним, что любая функция, вызываемая внутри `setTimeout` или `setInterval` вызывается в глобальном контексте, то есть `this` внутри таких функций будет равно `Window` или `undefined`:

```
const obj = {  
  value: 45,  
  getValue: function () { console.log('The value is ' + this.value); }  
};  
obj.getValue(); // The value is 45  
  
setTimeout(obj.getValue, 0); // The value is undefined
```

Arrow functions: this

```
function PrintLength(text) {  
  this.text = text;  
  this.getLength = function () {  
    console.log( 'The length is ' + this.text.length); }  
}
```

```
const t1 = new PrintLength('test me');  
t1.getLength(); // The length is 7
```

```
setTimeout(t1.getLength);
```

Uncaught TypeError: Cannot read property 'length' of undefined

Ошибка, так как функция запущена в глобальном контексте примерно так:

```
(function () {  
  console.log( 'The length is ' + this.text.length);  
})();
```

```
function PrintLength(text) {  
  this.text = text;  
  this.getLength = () => console.log( 'The length is ' +  
    this.text.length);  
}
```

```
const t1 = new PrintLength('test me');  
t1.getLength(); // The length is 7
```

```
setTimeout(t1.getLength);  
// The length is 7
```

Функция работала в ожидаемом режиме, так как `this` внутри стрелочной функции определяется в момент создания окружения для метода `this.getLength` - то есть он был равен экземпляру класса `PrintLength`

Arrow functions: constructors

Отсутствие “собственного” `this` у стрелочных функций не позволяет их использовать в качестве конструктора:

```
const User = () => console.log('test');  
new User();
```

Uncaught TypeError: User is not a constructor

Пример контекста и стрелочной функции можно посмотреть [здесь](#)

Arrow functions: constructors

Лексический `this` для arrow functions также не позволяет их использовать при создании методов конструктора (в прототипе):

```
function User(age) { this.age = age; }  
User.prototype.getAge1 = function () { return this.age; }  
User.prototype.getAge2 = () => return this.age;
```

```
const user = new User(25);
```

```
user.getAge1(); // 25
```

```
user.getAge2(); // undefined
```

Arrow functions: задачи

1. Переделать функцию с использованием функции-стрелки (в методе `reduce` тоже использовать arrow function):

```
function sum() {  
  const params = Array.prototype.slice.call(arguments);  
  if (!params.length) return 0;  
  return params.reduce(function (prev, next) { return prev + next; });  
}
```

```
sum(1, 2, 3, 4); // 10
```

```
sum(); // 0
```

Arrow functions: задачи

- Исправить функцию так, чтобы вместо `undefined` в массиве выводилось значение поля `prefix`:

```
const utils = {  
  numbers: [1,2,3,4],  
  prefix: 'number',  
  getOdd: function () {  
    return this.numbers.map(function (number) {  
      return this.prefix + ' - ' + number;  
    })  
  }  
};  
utils.getOdd(); // ["undefined - 1", "undefined - 2", "undefined - 3", "undefined - 4"]
```

Arrow functions: задачи

3. Переделать функции в стрелочные функции, использовать, где возможно, короткий синтаксис:

```
function getDate() { return new Date(); }
```

```
function getDay() {  
  const days = ['вс', 'пн', 'вт', 'ср', 'чет', 'пт', 'сб'];  
  return days[new Date().getDay()];  
}
```

```
function getListCopy(list) { return list.slice(); }
```