ES-2015

JavaScript

Переменные: let и const

Область видимости переменной let – блок {...}.

```
var a = 5;

if (true) {
  var a = 10;
}

alert(a); // 10
```

```
let a = 5;

if (true) {
  let a = 10;
  alert(a); // 10
}

alert(a); // 5
```

Переменная let видна только после объявления.

```
alert(a); // undefined

var a = 5;
```

```
alert(a); // ошибка, нет такой переменной

let a = 5;

let a = 10; // ошибка: переменная x уже объявлена
```

Объявление const задаёт константу

```
const a = 5;
a = 10; // ошибка
```

```
const user = {
  name: "Вася"
};

user.name = "Петя"; // допустимо
user = 5; // нельзя, будет ошибка
```

Деструктуризация

Деструктуризация (destructuring assignment) – это особый синтаксис присваивания, при котором можно присвоить массив или объект сразу нескольким переменным, разбив его на части.

Деструктуризация массива.

```
let [product1, product2] = ["lemon", "plum"];

alert(product1); // lemon
alert(product2);  // plum

// первый и второй элементы не нужны
let [, , title] = ["article 1", "article 2", "article 3", "article 4"];

alert(title); // article 3
```

Деструктуризация объекта

```
let options = {
  title: "Меню",
  width: 100,
  height: 200
};

let {title, width, height} = options;

alert(title);  // Меню
alert(width);  // 100
alert(height); // 200
```

Функции

Параметры по умолчанию

```
function showArticle(title = "Without title", width = 100, height = 200) {
  alert(title + ': ' + width + 'x' + height);
}

showArticle("ES-2015"); // ES-2015: 100x200

showArticle(undefined, null); //Without title: nullx200
```

Оператор spread вместо arguments

```
function showName(firstName, lastName, ...rest) {
  alert(firstName + ' ' + lastName + ' - ' + rest);
}

// выведет: Юлий Цезарь - Император,Рима
showName("Юлий", "Цезарь", "Император", "Рима");
```

Оператор spread при вызове

```
let numbers = [2, 3, 15];

// Оператор ... в вызове передаст массив как список аргументов
// Этот вызов аналогичен Math.max(2, 3, 15)
let max = Math.max(...numbers);

alert( max ); // 15
```

Деструктуризация в параметрах

```
let options = {
  title: "Меню",
  width: 100,
  height: 200
};

function showMenu({title, width, height}) {
  alert(title + ' ' + width + ' ' + height); // Меню 100 200
}

showMenu(options);
```

Функции в блоке

```
if (true) {

  sayHi(); // работает

  function sayHi() {
    alert("Привет!");
  }

}

sayHi(); // ошибка, функции не существует
```

Функции через =>

```
let inc = x => x+1;

let sum = (a,b) => a + b;

let getTime = () => {
  let date = new Date();
  let hours = date.getHours();
  let minutes = date.getMinutes();
  return hours + ':' + minutes;
};
```

Функции-стрелки не имеют своего this

```
let group = {
  title: "Наш курс",
  students: ["Вася", "Петя", "Даша"],

  showList: function() {
    this.students.forEach(
      student => alert(this.title + ': ' + student)
    )
  }
}

group.showList();
```

Строки

Строки-шаблоны

```
alert(`моя
  многострочная
  строка`);

let apples = 2;
let oranges = 3;

alert(`${apples} + ${oranges} = ${apples + oranges}`);
```

Полезные методы

- **str.includes(s)** – проверяет, включает ли одна строка в себя другую, возвращает true/false
- **str.endsWith(s)** – возвращает true, если строка str заканчивается подстрокой s.
- **str.startsWith(s)** – возвращает true, если строка str начинается со строки s.
- **str.repeat(times**) – повторяет строку str times раз.

Объекты и прототипы

Короткое свойство

```
let name = "Вася";
let isAdmin = true;

let user = {
  name,
  isAdmin
};
```

Методы объекта

```
let name = "Вася";
let user = {
  name,
  // вместо "sayHi: function() {...}" пишем "sayHi() {...}"
  sayHi() {
    alert(this.name);
  }
};

user.sayHi(); // Вася
```

super

```
let animal = {
  walk() {
    alert("I'm walking");
  }
};

let rabbit = {
  __proto__: animal,
  walk() {
    super.walk(); // I'm walking
  }
};
```

Классы

Class

```
class User {

  constructor(name) {
    this.name = name;
  }

  sayHi() {
    alert(this.name);
  }
}
```

Class Expression

```
let User = class {
  sayHi() { alert('Привет!'); }
};

new User().sayHi();
```

Геттеры и 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(' ');
  }
}
```

Геттеры и сеттеры

```
let user = new User("Вася", "Пупков");
alert( user.fullName );
user.fullName = "Иван Петров";
```

Статические свойства

```
class User {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  static createGuest() {
    return new User("Гость", "Сайта");
  }
};

let user = User.createGuest();
```

Наследование

```
class Rabbit extends Animal {
  walk() {
    super.walk();
    alert("...and jump!");
  }
}
```

Коллекции

Map

```
let map = new Map();

map.set('1', 'str1');
map.set(1, 'num1');
map.set(true, 'bool1');

// в обычном объекте это было бы одно и то же,
// map сохраняет тип ключа
alert( map.get(1)   ); // 'num1'
alert( map.get('1') ); // 'str1'

alert( map.size ); // 3
```

Итерация Map

```
for(let fruit of recipeMap.keys()) {
  alert(fruit);
}

for(let amount of recipeMap.values()) {
  alert(amount);
}

for(let entry of recipeMap) { // или recipeMap.entries()
  alert(entry);
}
```

Set

```
let set = new Set();

let vasya = {name: "Вася"};
let petya = {name: "Петя"};
let dasha = {name: "Даша"};

set.add(vasya);
set.add(petya);
set.add(dasha);
set.add(vasya);
set.add(petya);

// set сохраняет только уникальные значения
alert( set.size ); // 3
```

WeakMap и WeakSet

```
let activeUsers = [
  {name: "Вася"},
  {name: "Петя"},
  {name: "Маша"}
];

let weakMap = new WeakMap();

weakMap.set(activeUsers[0], 1);
weakMap.set(activeUsers[1], 2);
weakMap.set(activeUsers[2], 3);

alert( weakMap.get(activeUsers[0]) ); // 1
activeUsers.splice(0, 1); // Вася более не активный пользователь
// weakMap теперь содержит только 2 элемента
```

Promise

Promise – предоставляют удобный способ организации асинхронного кода.

Что такое Promise?

Promise – это специальный объект, который содержит своё состояние.

Он позволяет обрабатывать результаты асинхронных операций так, как если бы они были синхронными.

Promise может находиться в трёх состояниях:

- ожидание (pending): начальное состояние, не выполнено и не отклонено.
- выполнено (fulfilled): операция завершена успешно.
- отклонено (rejected): операция завершена с ошибкой.

Обработчики promise

- onFulfilled – срабатывают, когда promise в состоянии «выполнен успешно».

- onRejected – срабатывают, когда promise в состоянии «выполнен с ошибкой».

Создание promise

```
const myFirstPromise = new Promise((resolve, reject) => {
  // выполняется асинхронная операция, которая в итоге вызовет:
  //
  //   resolve(someValue); // успешное завершение
  // или
  //   reject("failure reason"); // неудача
});
```

Добавление обработчиков

```
promise.then(onFulfilled, onRejected)

promise.then(onFulfilled)

promise.then(null, onRejected)
```

.catch

```
promise.catch(onRejected)
```

throw

```
let p = new Promise((resolve, reject) => {
  // то же что reject(new Error("some error"))
  throw new Error("some error");
})

p.catch(alert); // Error: some error
```

Пример

```
function get(url) {
  return new Promise(function(resolve, reject) {
    var request = new XMLHttpRequest();
    request.open("GET", url, true);
    request.addEventListener("load", function() {
      if (request.status < 400)
        resolve(request.response);
      else
        reject(new Error("Request failed: " + request.statusText));
    });
    request.addEventListener("error", function() {
      fail(new Error("Network error"));
    });
    request.send();
  });
}
```

Пример

```
get("http://localhost:8080/users.json")
  .then(function(response) {
    console.log(response);
    return JSON.parse(response);
   })
  .then(function(data) {
    console.log(data[0]);
   })
  .catch(function(error){
    console.log("Error!!!");
    console.log(error);
   });
```

Модули

export

```
export let one = 1;

let two = 2;
export {two};

let three, four
export {three, four};
```

Экспорт функций и классов

```
export class User {
  constructor(name) {
    this.name = name;
  }
};

export function sayHi() {
  alert("Hello!");
};
```

import

```
import {one, two} from "./nums";

// импорт one под именем item1, а two - под именем item2
import {one as item1, two as item2} from "./nums";

//Импорт всех значений в виде объекта
import * as numbers from "./nums";
```