ООП. Классы.

В EcmaScript 6 по сравнению с EcmaScript 5 появятся классы.

EcmaScript 6:

// Supertype

class Person {

constructor(name) {

this.name = name;

}

describe() {

return "Person called " + this.name;

}

}

// Subtype

class Employee extends Person {

constructor(name, title) {

super.constructor(name);

this.title = title;

}

describe() {

return super.describe() + " (" + this.title + ")";

}

}

Теперь можно использовать эти классы:

let jane = new Employee("Jane", "CTO");

jane instanceof Person; // true

jane instanceof Employee; // true

jane.describe(); // 'Person called Jane (CTO)'

В EcmaScript 5 можно было добиться такого же результата с помощью прототипов:

// Supertype function Person(name) { this.name = name; }

Person.prototype.describe = function ()

{ return "Person called " + this.name; };

// Subtype

function Employee(name, title) { Person.call(this, name); this.title = title; } Employee.prototype = Object.create(Person.prototype);

Employee.prototype.constructor = Employee;

Employee.prototype.describe = function ()

{ return Person.prototype.describe.call(this) + " (" + this.title + ")";

};

**Сейчас, в большинстве современных языков существует такое понятие, как класс, которое ранее отсутствовала в javascript. Введение классов в js необходимо для упрощения работы программиста, для которого привычнее и удобнее работать именно с ними.**

2. Функции.

1) В EcmoScript 6 появятся, так называемые, arrow-функции. (стрелочные)

Пример:

let squares = [ 1, 2, 3 ].map(x => x \* x);

Реализация без стрелочных функций:

let squares = [ 1, 2, 3 ].map(function (x) { return x \* x });

Отличие от традиционных:

* Лексическое связывание. Значения *специальных переменных* **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 и кидают ошибку при использовании с оператором **new**.
* Недоступность «собственного» значения переменной **arguments**.

**Есть несколько причин для ввода стрелочных функций:**

**А) Связывание (binding) используется довольно часто в JavaScript и довольно легко потерять нужное значение this при использовании традиционных функций, что может привести к ошибкам.**

**Б) JS-движки смогут легко оптимизировать выполнение стрелочных функций за счет этих ограничений (в противоположность традиционным функциям, которые могут быть использованы в качестве конструктора и которые свободны для модификации специальных переменных).**

**В) Самая банальная причина: Такая форма записи просто короче.**