**Переход с ES5 на ES6**

**Особенности языка**

**Вступление**

Стандарт языка JavaScript ES6 вышел в 2016 году и произвел большие изменения в структуре языка и в подходе к программированию на JavaScript в целом. Вернулась блочная видимость, подход ООП, с его громоздкими объектами, был смещен функциональным программированием, использующим декларативный подход. Также появились константные и стрелочные функции и язык смог поддерживать более строгие конструкции.

**Возврат к блочной видимости переменных**

Появилось два новых ключевых слова для обозначения примитивов и объектов - let и const.

* Let – используется для создания объектов с возможностью дальнейшего изменения.
* Const – используется для обозначения переменных и объектов, с неизменяемой конструкцией.

Оба обозначения поддерживают блочную область видимости, то есть существуют только внутри контейнера, в котором были инициализированы. Контейнером может быть функция, цикл, метод и т. п.

Например:

const array = [1, 2, 3, 4];  
array = [5,6,7,8]; //Error  
let counts =[1,2,3,4];  
counts = [5,6,7,8]; //True construction

**Стрелочные функции**

Теперь в новом стандарте принято создавать функции как константы, чтобы их нельзя было переопределять. Конструкция использует стрелки (=>), поэтому и была названа таким образом.

Синтаксис:

const name = (arguments) => {

//тело функции

}

, где

* const /let – обозначаем неизменяемый или изменяемый объект соответственно.
* name – название функции.
* arguments – аргументы функции.

Например:

// Классическое функциональное выражение  
var addition = function (a, b) {  
 return a + b;  
};  
// Классическая стрелочная функция  
const trees = (a,b) => {  
 return a +b;  
}  
// Быстрая стрелочная функция  
const addition = (a, b) => a + b;

С этого момента функции и callBacks мы будем писать с помощью стрелочных функций. Посмотрим на пример стрелочной функции в forEach:

const array = [1, 2, 3, 4];  
let sum = 0;  
//Если аргумент всего один скобки - не обязательны  
//Если тело функции из одной строки, фигурные скобки можно опустить  
array.forEach((element => sum += element));  
console.log(sum); //10

**Значения по умолчанию**

Появилась возможность добавлять значения по умолчанию в функции. Делается это с помощью оператора “=” в списке аргументов функции.

Например:

//Напишем функцию возведения в степень с параметром по умолчанию  
const pow = (a, b = 2) => a \*\* b;  
//Вызов с одним аргументом  
console.log(pow(4)); //16  
//С двумя  
console.log(pow(4, 3)); //64

**Spread / Rest оператор**

“...“ - оператор называют как spread или rest, в зависимости от того, как и где он используется.

При использовании в любом перечисляемом объекте, данный оператор "разбивает" ("spread") его на индивидуальные элементы.

Например:

const console\_writer = (x, y, z) => {  
 console.log(x, y, z);  
}  
let arr = [1, 2, 3];  
console\_writer(...arr); // 1 2 3

Другим распространённым использованием оператора является объединение набора значений в один массив. В данном случае оператор добавляет к выполнению не только первый объект но и другие ("rest").

const console\_writer = (...args) => {  
 console.log(args);  
}  
console\_writer(1, 2, 3, 4, 5); // [1, 2, 3, 4, 5]

**Новые возможности литералов Object**

Теперь при создании объекта можно пропускать название свойства, если оно совпадает с названием переменной, которая это свойство записывает. Также появилась возможность динамически создавать названия свойств и методов.

Например:

const forestWriter = (name, count\_trees, tree\_name, area) => {  
 return {  
 name,  
 count\_trees,  
 tree\_name,  
 [name + "\_area"]: area  
 }  
};  
const forest = forestWriter(  
 "South",  
 132,  
 "fir",  
 "4000"  
);  
console.log(forest);  
/\*  
{  
 name: 'South',  
 count\_trees: 132,  
 tree\_name: 'fir',  
 South\_area: '4000'  
}   
\*/

**Деструктуризация массива**

Деструктуризация — это возможность присваивать значения массива поэлементно нескольким переменным сразу. Она помогает избежать использования вспомогательных переменных при взаимодействии с объектами и массивами.

Например:

const foo = () => {  
 return [1, 2, 3];  
}  
let arr = foo(); // [1,2,3]  
  
let [a, b, c] = foo();  
console.log(a); // 1  
console.log(b); // 2  
console.log(c); // 3

**Новой способ динамического создания строк**

ES6 предоставляет более простой способ вставки значения переменной или результата выражения, которые рассчитываются автоматически.

* ${ ... } используется для вычисления значения переменной/выражения.
* `` Обратные кавычки используются как разделитель для таких случаев (это которые над буквой ё).

Например:

let user = 'Дима';  
console.log(`Привет, ${user}!`); // Привет, Дима!

**Цикл for … in**

По структуре совпадает с for…of из ES5, но теперь в качестве аргумента он принимает не элемент, а индекс.

Например:

const array = [1, 2, 3];  
for (let index in array) {  
 console.log(array[index]);  
}  
/\*  
1  
2  
3  
\*/

**Новые структуры данных**

**Map и WeakMap**

Map имеет множество “имён”. Например, карта или словарь. Связаны они со спецификой этой структуры данных. Классический массив — это набор элементов, которые можно получить с помощью индекса. Если бы вместо индекса мы могли использовать кодовое слово или же адрес этого элемента, будто он находится на карте города, мы бы создали такую структуру данных, как Map. У каждого элемента есть key – ключ и value -значение (это может быть любой примитив или объект, как и в массиве). Так же каждый “словарь” имеет встроенные методы get и set, для получения и добавления элементов соответственно.

Синтаксис Map:

* const name = new Map – создание.
* name.set(key, value) – добавление значения (value) с привязкой ключа(key).
* name.get(key) – получение значения по ключу (key).
* name.has(value) – проверка на наличие значения (value) в данной карте, возвращает true или false.
* name.delete(key) – удаление значения по ключу (key).

Пример использования Map:

//Создадим Map  
const dictionary = new Map;  
  
const key = "forest\_name";  
const value = "South park"  
  
//Добавим первое значение  
dictionary.set(key, value);  
//Добавим второе значение  
dictionary.set("count\_trees", 352);  
  
//Выведем оба значения, получим значение по ключу  
dictionary.get(key);  
dictionary.get("count\_trees");

WeakMap аналог Map, но с одним правилом. Все ключи должны быть Object {}. Методы те же, что и у Map.

Пример той же программы, но с WeakMap:

//Создадим Map  
const dictionary = new WeakMap;  
  
const key = {name: "forest\_name"}; //теперь объект  
const value = "South park"  
  
//Добавим первое значение  
dictionary.set(key, value);  
//Добавим второе значение  
dictionary.set({count: "count\_trees"}, 352);  
  
//Выведем оба значения, получим значение по ключу  
dictionary.get(key);  
dictionary.get({count: "count\_trees"});

**Set и WeakSet**

Объекты Set это коллекции уникальных значений. Дублированные значения игнорируются, т.к. коллекция должна содержать только уникальные значения. Значения могут быть примитивами или ссылками на объекты.

Синтаксис:

* let mySet = new Set(arguments) – создание. arguments – список аргументов в массиве, например [1,2,3].
* name.add(value) – добавление значения (value).
* name.has(value) – проверка на наличие значения (value) в данном наборе, возвращает true или false.
* name.delete(value) – удаление значения (value).
* name.clear(value) – очищение коллекции.
* mySet.size – свойство, дающее размер Set.

Например:

let mySet = new Set([1, 1, 2, 2, 3, 3]);  
mySet.size; // 3, потому что копии объектов не сохраняются  
mySet.has(1); // true  
mySet.add('строки');  
mySet.add({a: 1, b: 2});  
  
console.log(mySet); //Set(5) { 1, 2, 3, 'строки', { a: 1, b: 2 } }  
  
mySet.clear();  
  
console.log(mySet);//Set(0) {}

Вы можете перебирать Set в цикле с помощью forEach или for...of. Перебор происходит в том же порядке, что и вставка.

**Классы в ES6**

Формат языка ES6 не добавляет классы как в языках, ориентированных на ООП, но делает синтаксис для прототипного наследования более “приятным” и наглядным для людей, предпочитающих объектно-ориентированный подход.

Данный раздел не будем разбирать досконально, так как синтаксис будет понятен любому ООП программисту.

Пример реализации класса с конструктором, свойством, динамическим и статическим методами:

class Forest {  
  
 constructor(name) {  
 console.log("Создан экземпляр forest!");  
 this.forest\_name = name; //добавление свойства прототипа  
 }  
  
 showName() {  
 console.log(this.forest\_name);  
 }  
  
 static loadAll() {  
 console.log("Рубим лес!");  
 }  
}  
  
let forestOne = new Forest("South park"); // "Создан экземпляр forest!"forestOne.showName(); *//South park*Forest.*loadAll*(); *//Рубим лес!*

Наследование и переопределение:

class Forest {  
 constructor(name) {  
 console.log("Создан экземпляр forest!");  
 this.forest\_name = name;   
 }  
 showName() {  
 console.log(this.forest\_name);  
 }  
 static loadAll() {  
 console.log("Рубим лес!");  
 }  
}  
  
class Park extends Forest { //extends команда для наследования  
 constructor(name) {  
 super(); //копирует конструктор родителя, обязательное поле  
 //Дополняем конструктор  
 this.park\_name = name;  
 console.log("Создан экземпляр park!");  
 }  
 //Переопределяем метод  
 showName() {  
 console.log(`${this.park\_name} но теперь это парк`);  
 }  
}  
  
const park = new Park("South park");  
//Создан экземпляр forest!  
// Создан экземпляр park!  
park.showName() //South park но теперь это парк