let gameRuleset = {

diceNumber: 1,

maxAttempts: 1

};

let firstCat = {

name: 'Кекс',

points: 0

};

let secondCat = {

name: 'Рудольф',

points: 0

};

let thirdCat = {

name: 'Рокки',

points: 0

};

let cats = [firstCat, secondCat, thirdCat];

let runGame = function (ruleset, players) {

for (let currentAttempt = 1; currentAttempt <= ruleset.maxAttempts; currentAttempt++) {

for (let i = 0; i < players.length; i++) {

let throwResult = keks.throwDice(ruleset.diceNumber, ruleset.diceNumber \* 6);

players[i].points += throwResult;

console.log(players[i].name + ' выбросил ' + players[i].points);

}

}

return players;

};

let getWinners = function (players) {

let winners = [];

let max = players[0];

for (let i = 0; i < players.length; i++) {

let currentPlayer = players[i];

if (currentPlayer.points > max.points) {

max = currentPlayer;

winners = [max];

} else {

if (currentPlayer.points === max.points) {

winners.push(currentPlayer);

}

}

}

return winners;

};

let printWinners = function (players, winners) {

if (players.length === winners.length) {

console.log('Все коты как на подбор!');

return;

}

let message = 'Победил ';

if (winners.length > 1) {

message = 'Победили ';

}

for (let i = 0; i < winners.length; i++) {

if (i >= 1) {

message += ', ';

}

message += winners[i].name;

}

message += ' с количеством очков: ' + winners[0].points;

console.log(message);

};

cats = runGame(gameRuleset, cats);

let tops = getWinners(cats);

printWinners(cats, tops);

**Конспект «Объекты». Раздел 1**

**Создание объекта**

Объект — тип данных, который хранит в себе информацию в виде пар «ключ-значение». Каждый элемент сопоставлен со своим ключом и порядок элементов совсем неважен.

Ключи в объекте следует делать уникальными. Если в одном объекте несколько одинаковых ключей, то используется значение последнего.

Несколько правил синтаксиса:

* Ключ обособляется от значения двоеточием.
* Пары «ключ-значение» отделяются друг от друга запятыми.
* Значениями могут быть данные любого типа (число, строка, массив и так далее).

let cat = {

name: 'Кекс',

age: 5

};

**Чтение из объекта**

Чтобы получить значение свойства, к нему надо обратиться через точку объект.ключ. Такая запись называется *точечной нотацией* и возвращает значение свойства объекта, если такое свойство есть. В противном случае вы получите undefined, то есть ничего.

console.log(cat.name); // Выведет в консоль: Кекс

console.log(cat.age); // Выведет в консоль: 5

console.log(cat.color); // Выведет: undefined. Такого ключа в объекте нет

**Запись в объект**

Свойства объектов можно не только читать, но и переопределять, как и обычные переменные. А ещё в объект можно добавлять новые свойства уже после того, как он был создан.

cat.age++; // Увеличили возраст кота на 1

console.log(cat.age) // Выведет: 6

cat.name = 'Рокки'; // Заменили снаружи значение свойства name

console.log(cat.name); // Выведет: Рокки

cat.color = 'рыжий'; // Добавили в объект новое свойство

console.log(cat.color); // Выведет: рыжий

**Передача по ссылке**

Объект всегда один, в памяти не создаётся новое место под копию объекта. Каждая переменная содержит не новую отдельную сущность, а ссылку на один-единственный объект. Поэтому когда мы меняем что-то в объекте через одну из переменных, в которой содержится ссылка на него, изменения видны во всех других переменных, будь их хоть двадцать или сорок. Это важная особенность объектов, которую надо запомнить. Она так и называется — *передача объектов по ссылке*.

let firstCat = {

name: 'Кекс',

age: 5

};

let secondCat = firstCat;

console.log(secondCat); // Выведет: {name: "Кекс", age:5}

firstCat.name = 'Снежок';

console.log(secondCat); // Выведет: {name: "Снежок", age:5}

let processorPrice = {

'i5': 5000,

'i7': 10000

};

let displayPrice = {

13: 5000,

15: 10000

};

let memoryPrice = {

8: 3000,

16: 4000

};

let buildComputer = function (memory, display, processor) {

let customComputer = {

basicPrice: 5000,

processor: processor,

display: display,

memory: memory,

getDescription: function () {

return 'компьютер с процессором ' + this.processor + ', диагональю ' + this.display + ', оперативной памятью ' + this.memory;

},

getPrice: function () {

return this.basicPrice + processorPrice[this.processor] + displayPrice[this.display] + memoryPrice[this.memory];

}

};

return customComputer;

};

let myComputer = buildComputer(8, 13, 'i7');

console.log('В корзине ' + myComputer.getDescription() + ' стоимостью ' + myComputer.getPrice());

let anotherComputer = buildComputer(16, 15, 'i5');

console.log('В корзине ' + anotherComputer.getDescription() + ' стоимостью ' + anotherComputer.getPrice());

# Конспект «Объекты». Раздел 2

## Методы объекта

В объектах могут храниться любые типы данных, в том числе и функции. Такие свойства-функции называются методами объектов. Вызов метода записывается так: объект.метод ().

let cat = {

name: 'Кекс',

color: 'рыжий',

age: 5,

getGreeting: function() {

return 'Мяу, привет!';

}

};

console.log(cat.getGreeting()); // Выведет: Мяу, привет!

## Скобочная нотация

Прочитать свойство из объекта можно с помощью квадратных скобок: catsFavoriteFood[name]. Способ со скобками называется *скобочной* нотацией, способ с точкой — *точечной* нотацией.

Скобочная нотация намного гибче точечной. Например, вы можете прочитать из объекта свойство, название которого записано в переменную:

let name = 'Кекс';

let catsFavoriteFood = { 'Кекс': 'рыба' };

console.log(catsFavoriteFood.name); // Выведет: undefined

console.log(catsFavoriteFood[name]); // Выведет: рыба

Первое сообщение содержит undefined, потому что у объекта нет свойства name. Второе сообщение содержит искомое значение, потому что программа понимает, что в квадратных скобках переменная. Значение переменной подставится в скобки, а затем будет найдено нужное свойство объекта.

В качестве ключей в объекте можно использовать любые строки, даже строки с пробелами. С точечной нотацией такие свойства прочитать не получится, а со скобочной — без проблем.

let cat = { 'favorite food': 'Сметана' };

// Вызовет ошибку

console.log(cat.favorite food);

// Отработает нормально

console.log(cat['favorite food']);

## Объект как словарь

Словари, или мапы, очень удобны в использовании. В нашем примере они хранят соотношение имени кота и лакомства, которое по вкусу именно ему.

let catsFavoriteFood = {

Кекс: 'рыба',

Рудольф: 'котлета',

Снежок: 'сметана'

};

let printFavoriteFood = function (name) {

// Используем скобочную нотацию

return 'Моя любимая еда — ' + **catsFavoriteFood[name]**;

};

console.log(printFavoriteFood('Снежок')); // Выведет: Моя любимая еда — сметана

## this

Изнутри методов можно обращаться к свойствам и другим методам объекта с помощью ключевого слова this. Оно указывает на текущий объект и называется *контекстом вызова*.

Важная деталь: пока функция не вызвана, this не содержит никакого значения, контекст появляется только в момент вызова функции.

let cat = {

name: 'Кекс',

color: 'рыжий',

age: 5,

getGreeting: function() {

return 'Мяу, привет! Меня зовут ' + this.name;

}

};

console.log(cat.getGreeting()); // Выведет: Мяу, привет! Меня зовут Кекс