**Лабораторная работа №0**

Командная оболочка МоngoDB представляет собой инструмент для администрирования базы и манипулирования данными с помощью команд на языке JavaScript. Команды в основном записываются в виде выражений языка JavaScript. Поэтому изучение МоngoDB разумно начать со знакомством с основами языка JavaScript.

**Язык JavaScript**

**Типы данных и переменные**

**1.Комментории**

В JavaScript комментарии начинаются с двух символов наклонной черты (//). Все, что идет следом за ними (в той же строке), интерпретатор JavaScript пропускает, поэтому комментарии не оказывают влияния на выполнение программы — это всего лишь пояснение. Комментарии могут занимать отдельную строку или следовать сразу после кода.

Другая разновидность комментариев это текст, записанный между /\* и \*/.

**2.Типы данных**

В JavaScript есть три основных типа данных: числа, строки и булевы значения.

В JavaScript числа записываются так:

5;

Любые текстовые данные записываются в строки. Строки выглядят так:

"Привет, я строка";

Булевы значения могут хранить одну из двух величин — либо это true («истина»), либо false («ложь»). Пример булева значения:

true;

Любые данные в JavaScript — сочетание этих основных типов.

**3.Числа и операторы**

JavaScript позволяет выполнять основные математические операции, такие как сложение, вычитание, умножение и деление. Для их записи используются символы +, −, \* и /, которые называют операторами**.**

1234 + 57 \* 3 - 31 / 4;

В математике существует правило, по которому умножение и деление выполняются прежде, чем сложение и вычитание, и JavaScript ему следует. Чтобы изменить порядок выполнения операций используются круглые скобки

8 / (1 + 3);

**4. Переменные**

Значениям в JavaScript можно давать имена, используя переменные. Чтобы создать новую переменную, используется ключевое слово var, после которого указывается имя переменной

var nick;.

Чтобы задать переменной значение, используется знак «равно»:

var age = 12;

Обновить значение переменной производится так

age = 15;

Можно создавать новые переменные, выполняя математические действия с переменными, созданными ранее.

**5. Инкремент и декремент**

Увеличение на 1 называют инкрементом, а уменьшение на 1 — декрементом. Выполняются инкремент и декремент с помощью операторов ++ и --.

var highFives = 0;

++highFives;

1

++highFives;

2

--highFives;

1

Чтобы изменить значение переменной на заданное число, можно написать такой код:

var score = 10;

score += 7;

17

score -= 3;

14

**6. Строки**

Чтобы создать строку, надо поставить знак двойной кавычки ("), затем ввести какой-нибудь текст и закрыть строку еще одной двойной кавычкой.

Строки можно хранить в переменных, так же как числа:

var myAwesomeString = "Что-то ОЧЕНЬ крутое!!!";

С помощью оператора + строки можно объединять: результатом будет новая строка, состоящая из первой строки, к концу которой присоединена вторая:

var greeting = "Привет";

var myName = "Ник";

greeting + myName;

"ПриветНик"

Чтобы узнать длину строки, достаточно добавить к ее концу .length:

"Суперпупердлиннаястрока".length;

23

Можно добавлять .length к концу как самой строки, так и переменной, содержащей строку.

Чтобы получить символ, стоящий в определенной позиции строки, используются квадратные скобки — [].

var myName = "Ник";

myName[0];

"Н"

myName[1];

"и"

myName[2];

"к"

Чтобы получить часть, или «срез», строки, используется slice.

Чтобы воспользоваться slice, ставится в конце строки (или переменной, содержащей строку) точка, а после нее слово slice и круглые скобки. В скобках указывается позиция первого символа той части строки, которую вы хотите получить, затем ставится запятая, а затем позиция последнего символа.

var longString = "Эта длинная строка такая длинная";

longString.slice (4, 18);

"длинная строка"

**7. Булевы значения**

Простое выражение с булевым значением:

var javascriptIsCool = true;

javascriptIsCool;

true

Булевы значения можно объединять посредством булевых (логических) операторов. Три основных булевых оператора — это && (и), || (или) и ! (не).

**8. Сравнение чисел**

Чтобы узнать, больше ли одно число, чем другое, нужно использовать оператор «больше» (>). Оператор, противоположный «больше» (>), зовется оператором «меньше» (<). Чтобы проверить два числа на точное равенство, используется тройной знак равенства (===). Есть еще один оператор сравнения (двойное равно, ==), который означает «практически равно».

**9. undefined и null**

В JavaScript есть два особых значения, они называются undeﬁned и null. Оба они означают «пусто», но смысл этого в обоих случаях немного различается. JavaScript использует значение undeﬁned, когда не может найти иного значения. А значение null обычно используется, чтобы явно обозначить — «тут пусто». Иначе говоря, null нужен, чтобы явно показать отсутствие значения.

**10.Массивы**

Чтобы создать массив, используйте квадратные скобки [].Чтобы создать массив со значениями, нужно перечислить эти значения внутри квадратных скобок, разделяя их запятыми. Отдельные значения, хранящиеся в массиве, называют элементами.

var dinosaurs = ["Тираннозавр", "Велоцираптор", "Стегозавр",

"Трицератопс", "Брахиозавр", "Птеранодон", "Апатозавр",

"Диплодок", "Компсогнат"];

Или иначе

var dinosaurs = [

"Тираннозавр",

"Велоцираптор",

"Стегозавр",

"Трицератопс",

"Брахиозавр",

"Птеранодон",

"Апатозавр",

"Диплодок",

"Компсогнат"

];

Чтобы ввести такой код в консоли, придется нажимать одновременно с ENTER клавишу SHIFT каждый раз, когда нужно перейти к новой строке. Иначе JavaScript попытается выполнить то, что уже введено, даже если команда еще не завершена. Поэтому при работе в консоли проще записывать массивы одной строкой.

Чтобы получить доступ к элементам массива, используются квадратные скобки с индексом нужного элемента, как в этом примере:

dinosaurs[0];

"Тираннозавр"

dinosaurs[3];

"Трицератопс"

Аналогично символам в строке, первому элементу массива соответствует индекс 0, второму — 1, третьему — 2 и т. д.

Чтобы заменить содержимое первого элемента массива dinosaurs ("Тираннозавр") на "Тираннозавр рекс", можно написать:

dinosaurs[0] = "Тираннозавр рекс";

Не обязательно, чтобы все элементы массива были одного типа.

Чтобы обратиться к элементам массива, вложенного в другой массив, нужно использовать вторую пару квадратных скобок. Например, если команда dinosaursAndNumbers[2]; вернет весь вложенный массив, то dinosaursAndNumbers[2][0]; — лишь первый элемент этого вложенного массива ("трицератопс").

dinosaursAndNumbers[2];

["трицератопс", "стегозавр", 3627.5]

dinosaursAndNumbers[2][0];

"трицератопс"

Порой нужно знать, сколько в массиве элементов. Для этого есть свойство length (длина), хранящее количество элементов в массиве. Чтобы узнать длину массива, просто надо добавить .length после его имени.

var maniacs = ["Якко", "Вакко", "Дот"];

maniacs[0];

"Якко"

maniacs[1];

"Вакко" maniacs[2];

"Дот"

maniacs.length;

3

Чтобы добавить элемент к концу массива, можно воспользоваться методом push. Надо ввести .push после имени массива, а после в круглых скобках указать элемент, который нужно добавить:

var animals = [];

animals.push("Кот");

1

animals.push("Пес");

2

animals.push("Лама");

3

animals;

["Кот", "Пес", "Лама"]

animals.length;

3

Чтобы добавить элемент в начало массива, используйте метод .unshift(элемент):

animals;

["Кот", "Пес", "Лама"]

animals[0];

"Кот"

animals.unshift("Мартышка");

4

animals;

["Мартышка", "Кот", "Пес", "Лама"]

animals.unshift("Белый медведь");

5

animals;

["Белый медведь", "Мартышка", "Кот", "Пес", "Лама"]

animals[0];

"Белый медведь"

animals[2];

"Кот"

Убрать из массива последний элемент можно, добавив к его имени .pop(). Метод pop делает сразу два дела: удаляет последний элемент из массива и возвращает этот элемент в виде значения.

animals;

["Белый медведь", "Мартышка", "Кот", "Пес", "Лама"]

var lastAnimal = animals.pop();

lastAnimal;

"Лама"

animals; ["Белый медведь", "Мартышка", "Кот", "Пес"]

animals.pop();

"Пес"

animals; ["Белый медведь", "Мартышка", "Кот"]

animals.unshift(lastAnimal);

4

animals;

["Лама", "Белый медведь", "Мартышка", "Кот"]

Чтобы «склеить» два массива, создав таким образом новый массив, используется команда ﬁrstArray. concat(otherArray). Метод concat создает массив, в котором элементы из ﬁ rstArray будут расположены перед элементами из otherArray.

var furryAnimals = ["Альпака", "Кольцехвостый лемур", "Йети"];

var scalyAnimals = ["Удав", "Годзилла"];

var furryAndScalyAnimals = furryAnimals.concat(scalyAnimals);

furryAndScalyAnimals;

["Альпака", "Кольцехвостый лемур", "Йети", "Удав", "Годзилла"] furryAnimals; ["Альпака", "Кольцехвостый лемур", "Йети"]

scalyAnimals;

["Удав", "Годзилла"]

Чтобы выяснить, какой у определенного элемента индекс в массиве, используется .indexOf("элемент").

var colors = ["красный", "зеленый", "синий"];

colors.indexOf("синий");

2

colors.indexOf("зеленый");

1

Воспользовавшись методом .join(), можно соединить все элементы массива в одну большую строку.

var boringAnimals = ["Мартышка", "Кот", "Рыба", "Ящерица"];

boringAnimals.join();

"Мартышка,Кот,Рыба,Ящерица"

**ОБЪЕКТЫ**

Объекты JavaScript очень похожи на массивы, но для доступа к элементам объектов используются строки, а не числа. Эти строки называют ключами, или свойствами, а элементы, которые им соответствуют, — значениями. Вместе эти фрагменты информации образуют пары «ключ-значение». Причем если массивы используются главным образом как списки, хранящие множество элементов, то объекты часто применяют как одиночные сущности с множеством характеристик, или атрибутов.

При создании объекта используются фигурные скобки {} . Внутри фигурных скобок можно вводить пары «ключ-значение», а вместе скобки и пары значений называются литералом объекта. Литерал объекта — это быстрый способ создания объекта вместе с его содержимым

var cat = {

"legs": 3,

"name": "Гармония",

"color": "Черепаховый"

};

Создана переменная под названием cat и присвоен ей объект с тремя парами «ключ-значение» (лапы, имя, окрас).

При создании объекта ключ записывается перед двоеточием (:), а значение — после. Это двоеточие напоминает знак «равно», поскольку значения, стоящие слева, присваиваются именам (ключам), стоящим справа, что похоже на создание переменных со значениями. Все пары «ключ-значение» должны быть разделены запятыми.

Создавая данный объект, имена ключей заключались в кавычках, однако это не обязательно. Следующая запись тоже является допустимым литералом объекта:

var cat = {

legs: 3,

name: "Гармония",

color: "Черепаховый"

};

JavaScript знает, что ключи всегда строковые, поэтому можно обходиться без кавычек. Имена ключей должны соответствовать тем же правилам, что и имена переменных: например, в них не должно быть пробелов. Но если ключ указан в кавычках, пробелы в его имени допустимы.Хотя ключ всегда является строковым (в кавычках он записан или без), значение, соответствующее этому ключу, может быть любого типа — даже переменной, в которой хранятся данные. Весь объект можно записать одной строкой, хотя читать такую программу будет, пожалуй, не слишком удобно:

var cat = { legs: 3, name: "Гармония", color: "Черепаховый" };

Хранящиеся в объектах значения можно получить с помощью квадратных скобок — так же, как элементы массива. Единственное различие в том, что вместо индек-са (число) используется ключ (строка).

cat["name"];

"Гармония"

Точно так же, как необязательны кавычки при записи литерала объекта, их можно опускать и при доступе к значениям по ключу. Однако в этом случае код будет немного другим:

cat.name;

"Гармония"

Такую запись называют точечной нотацией. Вместо того чтобы писать имя ключа в кавычках внутри квадратных скобок, мы просто ставим точку, после которой пишем имя ключа, без кавычек.

Чтобы узнать, какие вообще ключи есть у данного объекта, в JavaScript есть удобное средство — команда Object.keys():

var dog = { name: "Оладушек", age: 6, color: "белый",

bark: "Гав тяф тяф!" };

var cat = { name: "Гармония", age: 8, color: "черепаховый" };

Object.keys(dog);

["name", "age", "color", "bark"]

Object.keys(cat);

["name", "age", "color"]

Object.keys(anyObject) возвращает массив, содержащий все ключи объекта anyObject.

Пустой объект похож на пустой массив, только вместо квадратных скобок при его создании используются фигурные:

var object = {};

Добавлять элементы объекта можно так же, как элементы массива, — но используя строки вместо чисел:

var cat = {};

cat["legs"] = 3;

cat["name"] = "Гармония";

cat["color"] = "Черепаховый";

cat;

{ color: "Черепаховый", legs: 3, name: "Гармония" }

Новые ключи также можно добавлять через точечную нотацию.

var cat = {};

cat.legs = 3;

cat.name = "Гармония";

cat.color = "Черепаховый";

Можно сделать элементом массива или объекта другой массив или объект.

var dinosaurs = [

{ name: "Тираннозавр рекс", period: "Верхнемеловой" },

{ name: "Стегозавр", period: "Верхнеюрский" },

{ name: "Платеозавр", period: "Триасовый" }

];

Получить сведения о первом динозавре можно уже известным нам способом — указав индекс в квадратных скобках:

dinosaurs[0];

{ name: "Тираннозавр рекс", period: "Верхнемеловой" }

А если нужно только название первого динозавра, достаточно указать ключ объекта в еще одних квадратных скобках, следом за индексом:

dinosaurs[0]["name"];

"Тираннозавр рекс"

Другой вариант — воспользоваться точечной нотацией:

dinosaurs[1].period;

"Верхнеюрский"

Рассмотрим более сложный пример — массив объектов со сведениями о друзьях, где в каждый из объектов вложено по еще одному массиву. Сначала создадим объекты, а затем поместим их в массив.

var anna = { name: "Анна", age: 11, luckyNumbers: [2, 4, 8, 16] };

var dave = { name: "Дэйв", age: 5, luckyNumbers: [3, 9, 40] };

var kate = { name: "Кейт", age: 9, luckyNumbers: [1, 2, 3] };

Создно три объекта, сохранив их в переменных anna, dave и kate. У каждого из этих объектов есть по три свойства: name, age и luckyNumbers. Каждому ключу name соответствует строковое значение, ключу age — числовое, а ключу luckyNumbers — массив, содержащий несколько чисел. Теперь создадим массив друзей:

var friends = [anna, dave, kate];

Итак, в переменной friends находится массив с тремя элементами: anna, dave и kate (каждый из них является объектом). Можно получить любой из объектов по его индексу в массиве:

friends[1];

{ name: "Дэйв", age: 5, luckyNumbers: Array[3] }

Можно получить значение, хранящееся в объекте, указав индекс объекта в квадратных скобках, поставив точку и написав соответствующий ключ:

friends[2].name

"Кейт"

Можно даже получить значение из массива, находящегося в объекте, который, в свою очередь, находится в массиве friends:

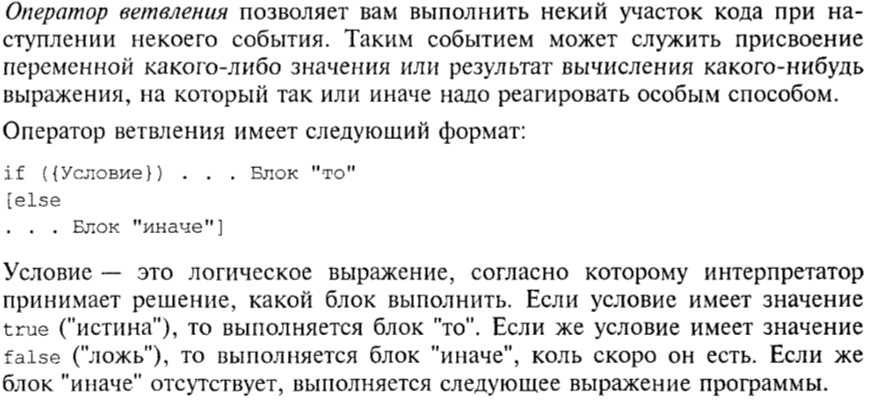
friends[0].luckyNumbers[1];

4

**УСЛОВИЯ И ЦИКЛЫ**

Условные конструкции и циклы — понятия, лежащие в основе любой мало-мальски серьезной программы. Их называют управляющими конструкциями, поскольку они позволяют решать, какие части кода и когда выполнять, а также насколько часто это нужно делать, исходя из заданных вами условий.

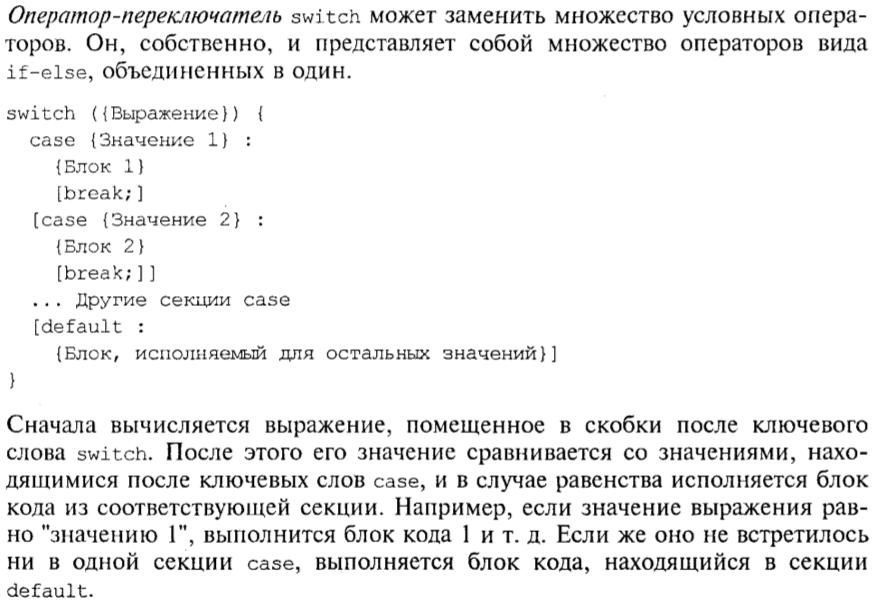
**Условные конструкции**

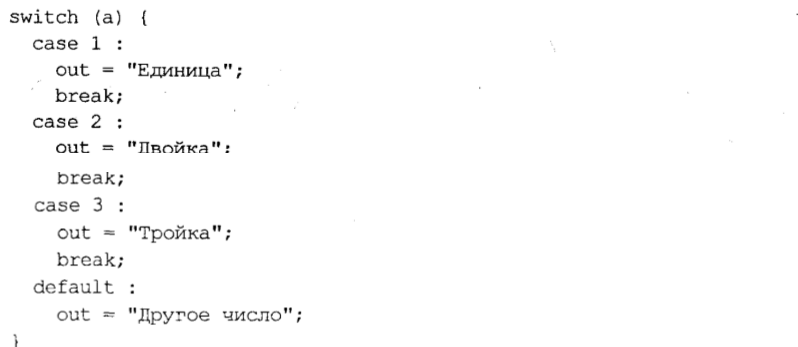
****

Пример.

****

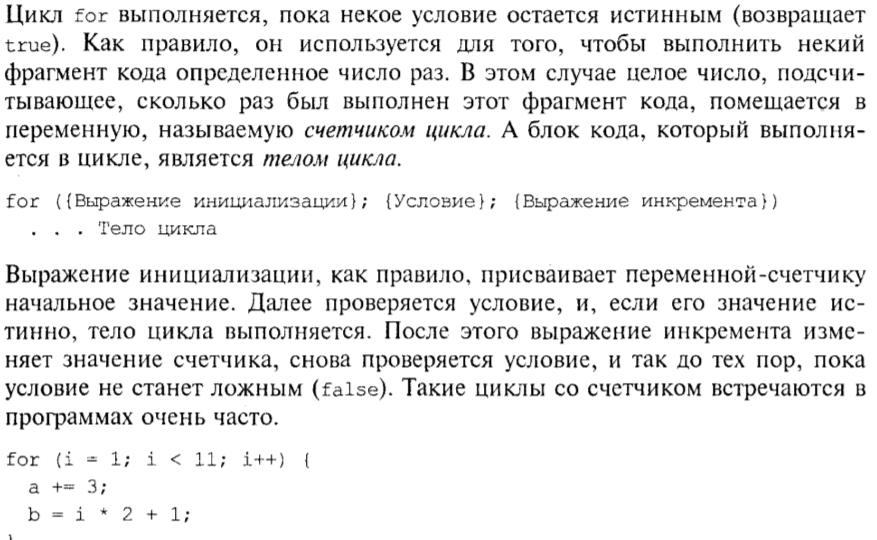
Обратите внимание на использование символов разрыва строки и пробела. Каждый программист волен самостоятельно выбирать наиболее удобный с его точки зрения стиль форматирования инструкций i f . . . e ls e . Вы могли видеть их записанными с использованием большего или меньшего количества строк или без пробелов между ключевыми словами и скобками. Все эти способы равноценны. Однако в любом случае следует всегда ставить во главу угла удобочитаемость кода, а не его компактность.



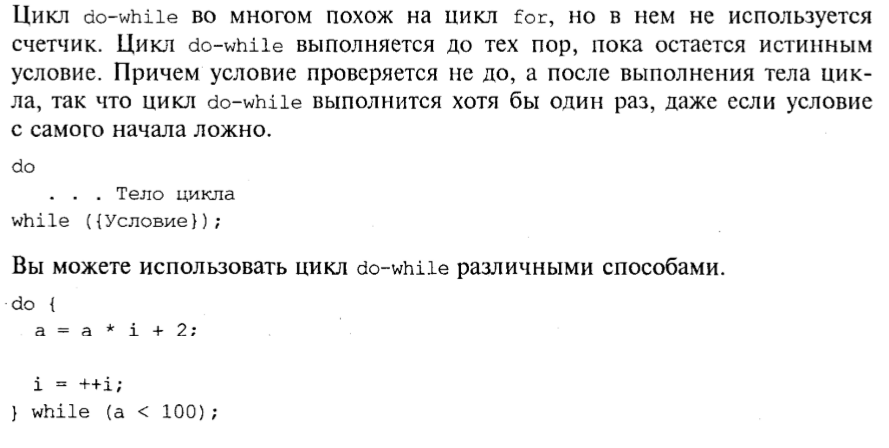
**Циклы**

Циклы, позволяют выполнять фрагмент кода многократно — до тех пор, пока выполняется некое условие.

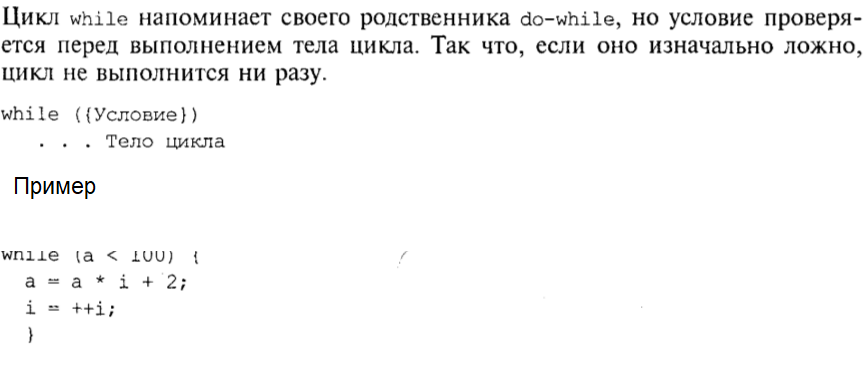
**for**

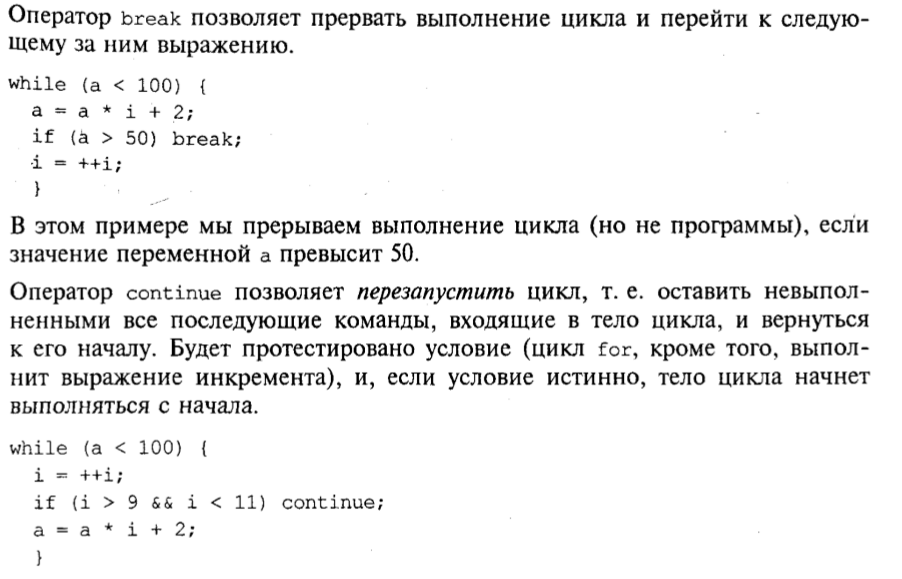


**do-while**

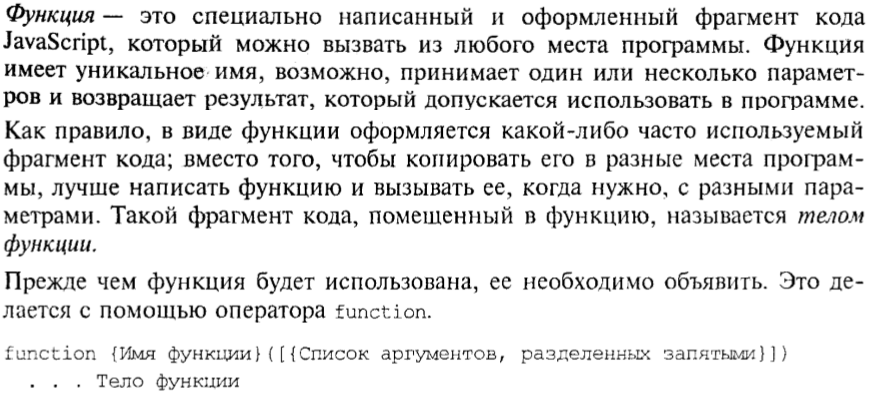


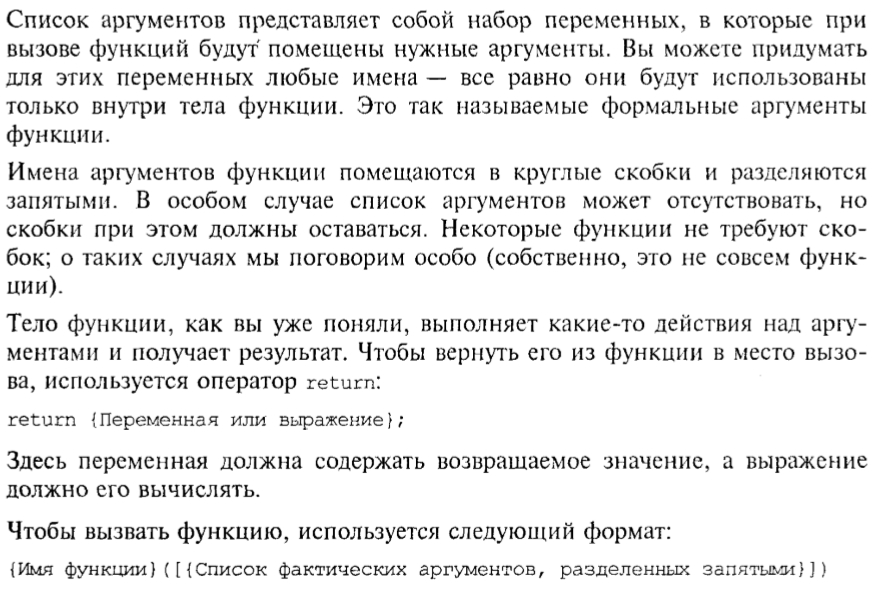
**While**

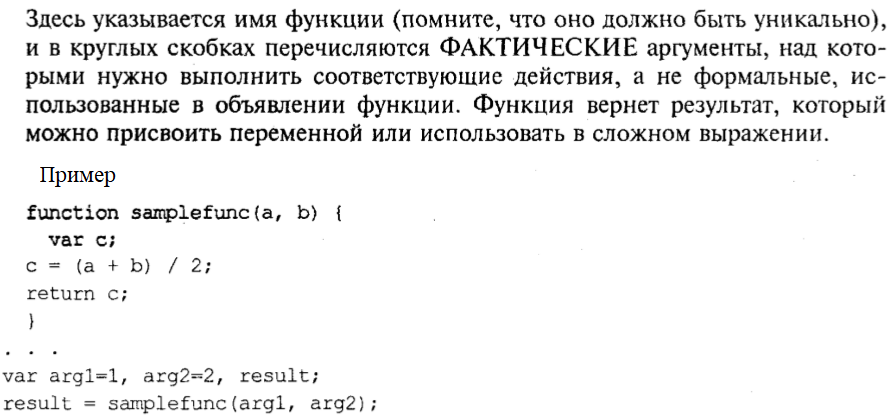


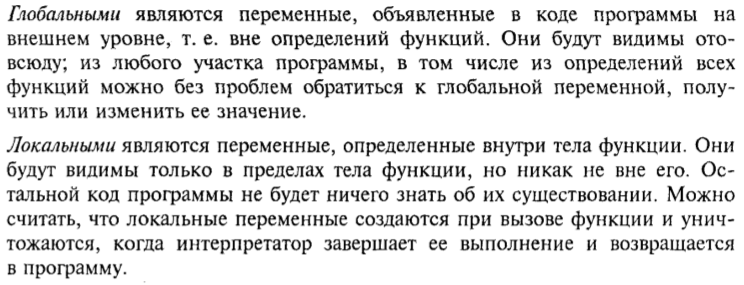


**Функции**

****







**Задание для самостоятельной работы**

Разработать презентацию, в которой была бы представлена информация о JavaScript, изложенная в данной инструкции.

**Содержание отчета**

**1**. Разработанная презентация