JavaScript. Основы

Основные понятия

- *JavaScript* — это язык программирования, позволяющий сделать Web-страницу интерактивной, то есть реагирующей на действия пользователя. Последовательность инструкций (называемая *программой*, *скриптом* или *сценарием*) выполняется *интерпретатором*, встроенным в обычный Web-браузер.

Основные понятия

- Иными словами, код программы внедряется в HTML-документ и выполняется на стороне клиента. Все программы выполняются в результате возникновения какого-то события. Например, перед отправкой данных формы можно проверить их на допустимые значения и, если значения не соответствуют ожидаемым, запретить отправку данных.

Основные понятия

- При изучении языков программирования принято начинать с программы, выводящей надпись "Hello, world":
- **<script type="text/javascript">**
- **<!--**
- **document.write("Hello, world");**
- **//-->**
- **</script>**

Основные понятия

- Строка
- **document.write("Hello, world");**
- содержащая инструкцию отобразить надпись "Hello, world" в окне Web-браузера, называется *выражением*. Каждое выражение в JavaScript заканчивается точкой с запятой.
- Все, что расположено после "//" до конца строки, в JavaScript считается *однострочным комментарием*:

Основные понятия

- Однострочный комментарий можно записать после выражения:
- **document.write("Hello, world"); // Однострочный комментарий**
- Кроме того, существует *многострочный комментарий*. Он начинается с символов /* и заканчивается символами */.

Ввод/вывод

- Предыдущий пример демонстрировал вывод данных в документ HTML.
- Рассмотрим другие способы вывода и ввода данных.
- **Окно с сообщением и кнопкой *OK***
- Метод **alert()** отображает диалоговое окно с сообщением и кнопкой **OK:**
- **window.alert("Hello, world");**

Ввод/вывод

- Сообщение можно разбить на строки с помощью последовательности символов \n.
- **Окно с сообщением и кнопками *OK* и *Cancel***
- Метод **confirm()** отображает диалоговое окно с сообщением и двумя кнопками **OK** и **Cancel**. Он возвращает значение **true**, если нажата кнопка **OK**, и false — если **Cancel**.

Ввод/вывод

- if (window.confirm("Нажмите одну из кнопок")) {
- window.alert("Нажата кнопка OK");
- }
- else { window.alert("Нажата кнопка C
- ancel"); }

Ввод/вывод

- **Окно с полем ввода и кнопками *OK* и *Cancel***
- Метод **prompt()** отображает диалоговое окно с сообщением, полем ввода и двумя кнопками **OK** и **Cancel**. Он возвращает введенное значение, если нажата кнопка **OK**, или специальное значение **null**, если нажата кнопка **Cancel**.

Ввод/вывод

- var n = window.prompt("Введите ваше имя", "Это значение по умолчанию");
- if (n==null) { document.write("Вы нажали Cancel"); }
- else { document.write("Привет " + n); }

Переменные

- *Переменные* — это участки памяти, используемые программой для хранения данных. Каждая переменная должна иметь уникальное имя в программе, состоящее из латинских букв, цифр и знаков подчеркивания. Первым символом может быть либо буква, либо знак подчеркивания. В имени переменной может также присутствовать символ $.

Переменные

- Имена переменных не должны совпадать с зарезервированными ключевыми словами языка JavaScript.
- Правильные имена переменных:
- **x, strName, y1, _name, frame1**
- Неправильные имена переменных:
- **1y, ИмяПеременной, frame**
- Последнее имя неправильное, так как является ключевым словом.

Переменные

- При указании имени переменной важно учитывать регистр букв: **strName** и **strname** — разные переменные.
- В программе переменные объявляются с помощью ключевого слова var. Можно объявить сразу несколько переменных в одной строке, указав их через запятую:
- **var x, strName, y1, _name, frame1;**

Определение типа данных

- В JavaScript переменные могут содержать следующие типы данных:
- **number** — целые числа или числа с плавающей точкой (дробные числа);
- **string** — строки;
- **boolean** — логический тип данных. Может содержать значения **true** (истина) или **false** (ложь);

Определение типа данных

- **function** — функции. В языке JavaScript ссылку на функцию можно присвоить какой-либо переменной. Для этого название функции указывается без круглых скобок. Кроме того, функции имеют свойства и методы;
- **object** — массивы, объекты, а также переменная со значением **null**.

Определение типа данных

- При инициализации переменной JavaScript автоматически относит переменную к одному из типов данных. Значение переменной присваивается с помощью оператора =.
- **Number1 = 7;**
- **Number2 = 7.8;**
- **String1 = "Строка";**
- **String2 = 'Строка';**
- **Boolean1 = true;**

Определение типа данных

- **Str1 = null; // Переменная Str1 пустая**
- Если в программе обратиться к переменной, которая не объявлена, то возникнет критическая ошибка. Если переменная объявлена, но ей не присвоено начальное значение, то значение предполагается равным **undefined**.
- Оператор **typeof** возвращает строку, описывающую тип данных переменной.

Операторы JavaScript

- Операторы позволяют выполнить определенные действия с данными. Операторы берут одно или два значения, представляющих собой переменную, константу или другое выражение, содержащее операторы или функции, и возвращают одно значение, определяемое по исходным данным. В основном они идентичны операторам языка С.

Операторы JavaScript

- **Математические операторы**
- **+** — сложение: **Z = X + Y;**
- **-** — вычитание: **Z = X - Y;**
- ***** — умножение: **Z = X * Y;**
- **/** — деление: **Z = X / Y;**
- **%** — деление по модулю: **Z = X % Y;**
- **++** — оператор инкремента. Увеличивает значение переменной на 1:

Операторы JavaScript

- **Z++; //Эквивалентно Z = Z + 1;**
- **--** — оператор декремента. Уменьшает значение переменной на 1:
- **Z--; //Эквивалентно Z = Z - 1;**
- Операторы инкремента и декремента могут использоваться в постфиксной или префиксной формах:
- **Z++; Z--; // Постфиксная форма**
- **++Z; --Z; // Префиксная форма**

Операторы JavaScript

- **Операторы присваивания**
- **=** присваивает переменной значение: **Z = 5;**
- **+=** увеличивает значение переменной на указанную величину:
- **Z += 5; // Эквивалентно Z = Z + 5;**
- **- =** уменьшает значение переменной на указанную величину:
- **Z - = 5; // Эквивалентно Z = Z - 5;**

Операторы JavaScript

- ***=** умножает значение переменной на указанную величину:
- **Z *= 5; // Эквивалентно Z = Z * 5;**
- **/=** делит значение переменной на указанную величину:
- **Z /= 5; // Эквивалентно Z = Z / 5;**
- **%=** делит значение переменной на указанную величину и возвращает остаток:
- **Z %= 5; // Эквивалентно Z = Z % 5;**

Операторы JavaScript

- **Двоичные операторы**
- **~** — двоичная инверсия: **Z = ~X;**
- **&** — двоичное И: **Z = X & Y;**
- **|** — двоичное ИЛИ: **Z = X | Y;**
- **^** — двоичное исключающее ИЛИ: **Z = X ^ Y;**
- **<<** — сдвиг влево на один или более разрядов с заполнением младших разрядов нулями: **Z = X << Y;**

Операторы JavaScript

- **>>** — сдвиг вправо на один или более разрядов с заполнением старших разрядов содержимым самого старшего разряда:
- **Z = X >> Y;**
- **>>>** — сдвиг вправо без учета знака на один или более разрядов с заполнением старших разрядов нулями:
- **Z = X >>> Y;**

Операторы JavaScript

- Двоичные операторы выполняют поразрядные действия с двоичным представлением целых чисел.
- **Оператор обработки строк**
- **+** — оператор конкатенации строк:
- **var Str = "Строка1" + "Строка2";// Str = "Строка1Строка2"**

Операторы JavaScript

- **Приоритет выполнения операторов**
- Перечислим операторы в порядке убывания приоритета:
- **!, ~, ++, --** — отрицание, двоичная инверсия, инкремент, декремент;
- *, /, % — умножение, деление, остаток от деления;
- **+, -** — сложение и вычитание;
- **<<, >>, >>>** — двоичные сдвиги;

Операторы JavaScript

- **&** — двоичное И;
- **^** — двоичное исключающее ИЛИ;
- **|** — двоичное ИЛИ;
- **=, +=, -=, *=, /=, %=** — присваивание.

Операторы JavaScript

- **Условные операторы.**
- Условные операторы позволяют в зависимости от значения логического выражения выполнить отдельный участок программы или наоборот не выполнять его. Логические выражения возвращают только два значения: **true** (истина) или **false** (ложь).

Операторы JavaScript

- **Операторы сравнения**
- Операторы сравнения используются в логических выражениях. Перечислим их:
- **==** — равно;
- **===** — строго равно;
- **!=** — не равно;
- **!==** — строго не равно;
- **<** — меньше; **>** — больше;

Операторы JavaScript

- **<=** — меньше или равно;
- **>=** — больше или равно.
- Отличие оператора **==** (равно) от оператора **===** (строго равно) в том, что если используется оператор **==**, интерпретатор пытается преобразовать разные типы данных к одному и затем сравнивает их. Оператор **===**, встретив данные разных типов, сразу возвращает **false** (ложь).

Операторы JavaScript

- Значение логического выражения можно инвертировать с помощью оператора **!** :
- **!(Var1 == Var2)**
- Несколько логических выражений можно объединить в одно :
- **&&** — логическое И;
- **||** — логическое ИЛИ.

Операторы JavaScript

- **Оператор ветвления *if…else*.**
- Аналогичен тому же из языка С.
- **Оператор ?**
- Оператор ? имеет следующий формат:
- **<Переменная> = (<Лог. выражение>) ? <если Истина> : <если Ложь>;**

Операторы JavaScript

- **Оператор выбора *switch*, операторы циклов *for, while, do…while, continue, break.***
- Синтаксис всех перечисленных операторов аналогичен тем же из языка С.

Преобразование типов данных

- Что будет, если к числу прибавить строку?
- **var Str = "5"; var Number1 = 3;**
- **var Str2 = Number1 + Str; // строка "35"**
- **var Str3 = Str + Number1; // строка "53"**
- В этом случае интерпретатор столкнется с несовместимостью типов данных и попытается преобразовать переменные к одному типу данных, а затем выполнить операцию.

Преобразование типов данных

- Переменная **Number1**, имеющая тип **number** (число), будет преобразована к типу **string** (строка), а затем будет произведена операция конкатенации строк.
- Другие операции:
- **var Number1 = 15;**
- **var Str = "5";**
- **var Str2 = Number1 - Str; // число 10**
- **var Str3 = Number1 * Str; // число 75**

Преобразование типов данных

- **var Str4 = Number1 / Str; // число 3**
- Интерпретатор попытается преобразовать строку в число, а затем вычислить выражение. Не важно, в какой последовательности будут указаны число и строка:
- **var Str5 = Str * Number1; // Переменная содержит число 75**
- Если в строке будут одни буквы:

Преобразование типов данных

- **var Number1 = 15;**
- **var Str = "Строка";**
- **var Str2 = Number1 - Str; // Переменная содержит значение NaN**
- В этом случае интерпретатор не сможет преобразовать строку в число и присвоит переменной значение **NaN** (Not a Number, не число).

Преобразование типов данных

- Хорошо, что интерпретатор сам делает преобразование типов данных. Но можно получить результат, который не планировался. По этой причине лучше оперировать переменными одного типа, а если необходимо делать преобразования типов, то делать это самим, используя следующие встроенные функции JavaScript:

Преобразование типов данных

- **parseInt**(<Строка>, [<Основание>]) преобразует строку в целое число. Строка считается заданной в системе счисления, указанной вторым необязательным параметром. Если основание не указано, то по умолчанию используется десятичная система. Если строка не может быть преобразована в число, возвращается значение NaN.

Преобразование типов данных

- **parseFloat**(<Строка>) преобразует строку в число с плавающей точкой
- **eval**(<Строка>) вычисляет выражение в строке, как будто это было обычное выражение JavaScript:
- **var Str = "3 + 5";**
- **var Str2 = eval(Str); // Переменная содержит число 8**

Массивы

- *Массив* — это нумерованный набор перемен-ных. Переменная в массиве называется *эле-ментом* массива, а ее позиция в массиве - *индексом*. Нумерация элементов массива начинается с 0. Количество элемен-тов в массиве называется *размером* массива.
- При инициализации массива переменные указываются через запятую в квадратных скобках:

Массивы

- **Mass1 = [1, 2, 3, 4];**
- Получить значение элемента массива можно, указав его индекс в квадратных скобках:
- **Str = Mass1[0]; // Переменной Str будет присвоено значение 1**
- При желании можно добавить новый элемент массива или изменить значение существующего:

Массивы

- **Mass1[5] = 6; Mass1[0] = 0;**
- Любому элементу массива можно присвоить другой массив: **Mass1[0] = [1, 2, 3, 4];**
- Следует учитывать, что операция присваивания сохраняет в переменной ссылку на массив, а не все его значения.
- Чтобы сделать копию массива, можно, например, воспользоваться методом **slice()**, который возвращает срез массива:

Массивы

- **var Mass1, Mass2; Mass1 = [1, 2, 3, 4];**
- **Mass2 = Mass1.slice(0);**
- **Mass2[0] = "Новое значение";**
- **document.write(Mass1.join(", ") + "
");**
- **document.write(Mass2.join(", "));**
- Результат:
- **1, 2, 3, 4**
- **Новое значение, 2, 3, 4**

Массивы

- При использовании многомерных массивов метод **slice()** создает "поверхностную" копию, а не полную:
- **var Mass1, Mass2; Mass1 = [[0, 1], 2, 3, 4];**
- **Mass2 = Mass1.slice(0);**
- **Mass2[0][0] = "Новое значение1";**
- **Mass2[1] = "Новое значение2";**

Массивы

- В результате массивы будут выглядеть так:
- **Mass1 = [["Новое значение1", 1], 2, 3, 4];**
- **Mass2 = [["Новое значение1", 1], "Новое значение2", 3, 4];**

Функции

- *Функция* — это фрагмент кода JavaScript, который можно вызвать из любого места программы. Функция описывается с помощью ключевого слова **function** по следующей схеме:
- **function <Имя функции> ([<Параметры>]) {**
- **<Тело функции>**
- **[return <Значение>]**
- **}**

Функции

- Функция должна иметь уникальное имя. Для имен действуют такие же правила, что и для переменных. После имени функции в круглых скобках можно указать один или несколько параметров через запятую. Параметров может вообще не быть. В этом случае указываются только круглые скобки. Между фигурными скобками располагаются выражения JavaScript.

Функции

- Функция может возвращать значение в место вызова функции с помощью ключевого слова **return**.
- Пример функции без параметров:
- **function f_Alert_OK() {**
- **window.alert("Сообщение при удачно выполненной операции");**
- **}**

Функции

- Пример функции с параметром:
- **function f_Alert(msg) {**
- **window.alert(msg); }**
- В качестве возвращаемого значения в конструкции **return** можно указывать не только имя переменной, но и выражение:
- **function f_Sum(x, y) {**
- **return (x + y); }**

Функции

- Вызов функции в программе :
- **f_Alert_OK(); f_Alert("Сообщение");**
- **Var1 = f_Sum(5, 2); // Var1 = 7**
- Выражения, указанные после **return <значение>;**, никогда не будут выполнены:
- **function f_Sum(x, y) {**
- **return (x + y); window.alert("Сообщение");**
- **// Это выражение не будет выполнено**
- **}**

Функции

- Имя переменной, передающей значение функции, может не совпадать с именем переменной внутри функции:
- **function f_Sum(x, y) {**
- **return (x + y); }**
- **var Var3, Var1 = 5; var Var2 = 2;**
- **Var3 = f_Sum (Var1, Var2);**

Функции

- Ссылку на функцию можно сохранить в какой-либо переменной. Для этого название функции указывается без круглых скобок:
- **function test() {**
- **window.alert("Это функция test()"); }**
- **var x = test; // ссылка на функцию**
- **x(); // Вызов test() через переменную x**

Функции

- Функция может вообще не иметь названия. В этом случае ссылку на анонимную функцию сохраняют в переменной:
- **var x = function() { // ссылка**
- **window.alert("Сообщение"); };**
- **x(); // Вызов анонимной функции через x**
- Ссылку на вложенную функцию можно вернуть в качестве значения в инструкции **return**.

Функции

- Чтобы вызвать вложенную функцию, круглые скобки указываются два раза:
- **var x = function() { // ссылка**
- **return function() { // Возврат ссылки**
- **window.alert("Это вложенная функция");**
- **}; };**
- **x()(); // Вызов вложенной функции через x.**

Ошибки времени выполнения

- Это ошибки, которые возникают во время работы скрипта из-за событий, не предусмотренных программистом.
- Попытка деления на ноль возвращает Infinity:
- window.alert(5/0); // Infinity
- При обращении к несуществующему элементу массива возвращается значение undefined: var arr = [1, 2];
- window.alert(arr[20]); // undefined

Обработка ошибок

- Перехватить и обработать ошибки позволяет конструкция **try/catch/finally**.
- Конструкция имеет следующий формат:
- **try {**
- **<Выражения, в которых перехватываем ошибки> }**
- **[catch ([<Ссылка на объект Error>]) {**
- **<Обработка ошибки> }]**

Обработка ошибок

- **[finally {**
- **<Выражения, которые будут выполнены в любом случае> }]**
- Выражения, в которых могут возникнуть ошибки, размещаются в блоке **try**. Если внутри этого блока возникнет исключение, то управление будет передано в блок **catch**. В качестве параметра в блоке **catch** можно указать переменную, через которую будет доступен объект **Error**, содержащий описание ошибки.

Обработка ошибок

- Если в блоке **try** ошибки не возникло, то блок **catch** не выполняется. Если указан блок **finally**, то выражения внутри этого блока будут выполнены независимо от того, возникла ошибка или нет. Блоки **catch** и **finally** являются необязательными, но хотя бы один из них должен быть указан.

Обработка ошибок

- В некоторых случаях требуется не обрабатывать ошибку, а, наоборот, указать программе, что возникла неисправимая ошибка, и прервать выполнение всей программы. Для этого предназначен оператор **throw**:
- **if (d < 0)**
- **throw new Error("Переменная не может быть меньше нуля");**

Встроенные классы JavaScript

- *Класс* — это тип объекта, включающий в себя переменные и функции для управления этими переменными. Переменные называют *свойствами*, а функции — *методами*.
- В JavaScript есть встроенные классы:
- ***Global, Number, String, Array, Math, Date, Function, Arguments, RegExp.***
- ***Рассматривать их не будем.***

События

- При взаимодействии пользователя с Web-страницей происходят события, извещения системы о том, что пользователь выполнил какое-либо действие или внутри самой системы возникло некоторое условие. События возникают при щелчке на элементе, перемещении мыши, нажатии клавиши на клавиатуре, изменении размеров окна, окончании загрузки Web-страницы и т. д.
- Названия событий начинаются с префикса **on**.

События

- **События мыши**
- **onmousedown** — при нажатии кнопки мыши на элементе Web-страницы или самой странице;
- **onmouseup** — при отпускании ранее нажатой кнопки мыши;
- **onclick** — при щелчке мыши на элементе Web-страницы или на самой Web-странице;
- **ondblclick** — при двойном щелчке мыши;

События

- **onmousemove** — при перемещении мыши;
- **onmouseover** — при наведении курсора мыши на элемент Web-страницы;
- **onmouseout** — при убирании курсора мыши с элемента Web-страницы;
- **onselectstart** — при начале выделения текста;
- **onselect** — при выделении элемента;
- **oncontextmenu** — при нажатии правой кнопки мыши для вывода контекстного меню.

События

- **События клавиатуры**
- **onkeydown** — при нажатии клавиши на клавиатуре;
- **onkeypress** — аналогично **onkeydown**, но возвращает значение кода символа в кодировке Unicode. Наступает постоянно, пока пользователь не отпустит клавишу;
- **onkeyup** — при отпускании ранее нажатой клавиши клавиатуры;

События

- **onhelp** — при нажатии клавиши <F1>.
- **События документа**
- **onload** — после загрузки Web-страницы;
- **onscroll** — при прокручивании содержимого элемента страницы, документа, окна или фрейма;
- **onresize** — при изменении размеров окна;
- **onbeforeunload** — перед выгрузкой документа;

События

- **onunload** — непосредственно перед выгрузкой документа. Наступает после события **onbeforeunload**;
- **onbeforeprint** — перед распечаткой документа или вывода его на предварительный просмотр;
- **onafterprint** — после распечатки документа или вывода его на предварительный просмотр.

События

- **События формы**
- **onsubmit** — при отправке данных формы;
- **onreset** — при очистке формы;
- **onblur** — при потере фокуса элементом формы;
- **onchange** — при изменении данных в текстовом поле и перемещении фокуса на другой элемент формы (наступает перед событием **onblur**);

События

- **onfocus** — при получении фокуса элементом формы.
- События последовательно передаются элементу-родителю. Такое прохождение событий называется всплыванием событий.
- Если всплывание событий необходимо прервать, свойству **cancelBubble** объекта **event** следует присвоить значение **true**.