Основы языка JavaScript

JavaScript появился благодаря усилиям двух компаний - Netscape (Mozilla) и Sun Microsystems (Oracle).

JavaScript позволяет создавать приложения, выполняемые как на стороне клиента, т.е. эти приложения выполняются браузером на компьютере пользователя, так и на стороне сервера.

Что такое JavaScript?

- Изначально *JavaScript* был создан, чтобы *«сделать веб-страницы живыми»*.
- Программы на этом языке называются *скриптами*. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы.
- Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска. Это отличает JavaScript от другого языка – Java.

В браузере JavaScript (оживление страницы) может:

Добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили.

Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.

Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы

задавать вопросы посетителю, показывать сообщения.

Запоминать данные на стороне клиента («local storage»).

Возможности JavaScript:

- ✔создание динамических страниц, реагирующих на действия пользователя;
- ✔обработка элементов форм в режиме реального времени (проверка правильности ввода данных)
- ✔создание полноценных приложений, работающих в пределах сайта
- ✔отслеживание действий, совершаемых пользователями и др.

Что делает JavaScript особенным?

***Три* сильные стороны JavaScript:**

Полная интеграция с HTML/CSS.

Простые вещи делаются просто.

Поддерживается всеми основными браузерами и включён по умолчанию.

JavaScript – это единственная браузерная технология, сочетающая в себе все эти три вещи.

Способы размещения JavaScript кода

1. включение кодов JavaScript между тэгами <SCRIPT> и </SCRIPT>;
2. подключение внешнего файла с кодами JavaScript с помощью тэга <SCRIPT>;

Тег «script»

```
<!DOCTYPE HTML>
<html>

<body>

  <p>Перед скриптом...</p>

  <script>
    alert( 'Привет, мир!' );
  </script>

  <p>...После скрипта.</p>

</body>

</html>
```

Внешние скрипты

- Если у вас много JavaScript-кода, вы можете поместить его в отдельный файл.

```
<script src="/path/to/script.js"></script>
```

Для подключения нескольких скриптов используйте несколько тегов:

```
<script src="/js/script1.js"></script>
<script src="/js/script2.js"></script>
```

1. Включение JavaScript между тэгами <SCRIPT> и </SCRIPT>

Для включения фрагментов программы на JavaScript или другом скрипте (объявлений переменных, описаний функций, операторов, вызовов функций и др.) обычно используется следующий шаблон:

<**SCRIPT** [**language**="Язык программирования, на котором написан скрипт"] [**src**="Адрес файла со скриптом"]>

программный код JavaScript

</**SCRIPT**>

Если при разработке сценария используется язык JavaScript, то параметр **language** можно не указывать.

СТРУКТУРА КОДА

- **Инструкции**
- Инструкции – это синтаксические конструкции и команды, которые выполняют действия.

`alert('Привет, мир!')`, которая отображает сообщение «Привет, мир!».

- **Точка с запятой**
  - В большинстве случаев точку с запятой можно не ставить, если есть переход на новую строку.
  - Так тоже будет работать:

```
alert('Привет')
alert('Мир')
```

  - перенос строки как «неявную» точку с запятой. Это называется автоматическая вставка точки с запятой.

```
alert(3 +
1
+ 2);
```

JavaScript «забывает» вставить точку с запятой там, где она нужна.

- **Пример ошибки**

```
alert("Сейчас будет ошибка")

[1, 2].forEach(alert)
```

Теперь, если запустить код, выведется только первый `alert`, а затем мы получим ошибку!
Всё исправится, если мы поставим точку с запятой после `alert`

2. Использование точки с запятой для указания конца инструкции не является обязательным условием. В JavaScript между инструкциями можно не ставить точку с запятой, если они находятся на разных строках.

первая инструкция

вторая инструкция

3. При размещении инструкций на одной строке, их надо обязательно разделять с помощью точки с запятой, тем самым сообщая интерпретатору, где заканчивается первая инструкция и начинается следующая.

первая инструкция; вторая инструкция;

4. Хорошей практикой в программировании является использование точки с запятой всегда, даже если инструкции расположены на разных строках.

Это поможет сделать ваш код более читабельным и возможно избежать в дальнейшем непредвиденных ошибок во время исполнения программы.

Комментарии

Комментарии – пояснения к исходному коду программы, оформленные по правилам, определённым языком программирования.

Функции комментариев:

- ✔помогают правильно понять текст программы;
- ✔временное исключение части кода программы.

Виды комментариев:

- ✔однострочный;

```
// это однострочный комментарий
```

- ✔многострочный.

```
/*  Это многострочный комментарий. Он
расположен на нескольких строках */
```

КОММЕНТАРИИ

Однострочные комментарии начинаются с двойной косой черты //.

```
// Этот комментарий занимает всю строку
alert('Привет');

alert('Мир'); // Этот комментарий следует за инструкцией
```

Многострочные комментарии начинаются косой чертой со звёздочкой /* и заканчиваются звёздочкой с косой чертой */.

```
/* Пример с двумя сообщениями.
Это - многострочный комментарий.
*/
alert('Привет');
alert('Мир');
```

Правила объявления переменных

Переменная – это именованная область в оперативной памяти компьютера, предназначенная для хранения различной информации.

Для того, чтобы использовать переменную, ее сначала необходимо *объявить*: написать перед ее именем ключевое слово let. Давайте объявим, например, переменную с именем a:

```
let a;
```

```
let a = 3; // объявляем переменную и задаем ей
  значение
alert(a);  // выведет 3
```

```
let a; // объявим переменную
a = 3; // присвоим ей значение
alert(a); // выведем значение переменной на экран
```

Объявление нескольких переменных в одной строке:

```
let user = 'John', age = 25, message = 'Hello';
```

Для лучшей читаемости объявляйте каждую переменную на новой строке.

```
let user = 'John';
let age = 25;
let message = 'Hello';
```

Var вместо let

В старых скриптах вы также можете найти другое ключевое слово: `var` вместо `let`:

```
var message = 'Hello';
```

Ключевое слово `var` – *почти* то же самое, что и `let`. Оно объявляет переменную, но немного по-другому, «устаревшим» способом.

- переменную message можно представить как коробку с названием "message" и значением "Hello!" внутри:

```
let message;

message = 'Hello!';

message = 'World!'; // значение изменено

alert(message);
```

Мы также можем изменить его столько раз, сколько захотим:

- Мы также можем объявить две переменные и скопировать данные из одной в другую.

```
let hello = 'Hello world!';

let message;

// копируем значение 'Hello world' из переменной hello в переменную message
message = hello;

// теперь две переменные содержат одинаковые данные
alert(hello); // Hello world!
alert(message); // Hello world!
```

Переменная может быть объявлена только один раз.
Повторное объявление той же переменной является ошибкой:

Имена переменных.

1. Имя переменной должно содержать только буквы, цифры или символы $ и _.
2. Первый символ не должен быть цифрой.
3. Если имя содержит несколько слов, (слова следуют одно за другим), то каждое следующее слово начинается с заглавной буквы: myVeryLongName.
4. Существует список зарезервированных слов, которые нельзя использовать в качестве имён.

Регистр имеет значение

Переменные с именами `apple` и `AppLE` – это две разные переменные.

Примеры допустимых идентификаторов

```
let userName;
let test123;
```

```
let $ = 1; //
let _ = 2; //
alert($ + _);
```

```
let 1a;
```

```
let my-name;
```

```
let return = 5;
```

```
let let = 5;
```

Зарезервированные слова

JavaScript резервирует ряд идентификаторов, которые играют роль ключевых слов самого языка.

break	delete	function	return	typeof
case	do	if	switch	var
catch	else	in	this	void
continue	false	instanceof	throw	while
debugger	finally	new	true	with
default	for	null	try	

КОНСТАНТА

константа – запись в исходном коде программы, представляющая собой обычное фиксированное значение.

```
const myBirthday = '18.04.1982';
```

Переменные, объявленные с помощью const, называются «константами». Их нельзя изменить. Попытка сделать это приведёт к ошибке:

```
const myBirthday = '18.04.1982';

myBirthday = '01.01.2001'; // ошибка, константу нельзя перезаписать!
```

Константы в верхнем регистре

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
```

Типы данных JavaScript

Типы данных в JavaScript делятся на две категории:

- ✔простые (примитивные) типы;
- ✔составные (объекты).

К категории простых типов относятся:

- ✔**String** - текстовые строки (строки)
- ✔**Number** - числа
- ✔**Boolean** - логические (булевы) значения
- ✔**null**
- ✔**undefined**

К составным типам данных относятся:

- ✔**Function** - функции
- ✔**Array** - массивы
- ✔**Object** - объекты

Числа

Числовой тип данных (number)

```
let n = 123;
n = 12.345;
```

Существует множество операций для чисел, например, умножение *, деление /, сложение +, вычитание - и так далее.

Тип BigInt был добавлен в JavaScript, чтобы дать возможность работать с целыми числами произвольной длины.

$(2^{53}-1)$ (т. е. 9007199254740991),

```
// символ "n" в конце означает, что это BigInt
const bigInt = 1234567890123456789012345678901234567890n;
```

Специальные числовые значения

В JavaScript имеются предопределённые глобальные переменные **Infinity** и **NaN**.

Переменная **Infinity** хранит специальное значение обозначающее бесконечность.

```
alert( 1 / 0 ); // Infinity
```

Переменная **NaN** также хранит специальное значение NaN (- не число).

```
alert( "не число" / 2 );
```

Строки

Строка (string) в JavaScript должна быть заключена в кавычки.

```
let str = "Привет";
let str2 = 'Одинарные кавычки тоже подойдут';
let phrase = `Обратные кавычки позволяют встраивать переменные ${str}`;
```

В JavaScript существует три типа кавычек.

1. Двойные кавычки: "Привет".
2. Одинарные кавычки: 'Привет'.
3. Обратные кавычки: `Привет`.

Обратные кавычки имеют расширенную функциональность. Они позволяют нам встраивать выражения в строку, заключая их в

```
let name = "Иван";

// Вставим переменную
alert( `Привет, ${name}!` ); // Привет, Иван!

// Вставим выражение
alert( `результат: ${1 + 2}` ); // результат: 3
```

Булевый тип

- Булевый тип (boolean) может принимать только два значения:

true (истина) и false (ложь).

```
let nameFieldChecked = true; // да, поле отмечено
let ageFieldChecked = false; // нет, поле не отмечено
```

```
let isGreater = 4 > 1;

alert( isGreater ); // true (результатом сравнения будет "да")
```

Значение «Значение «null»

```
let age = null;
```

- Это просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно».
- В приведённом выше коде указано, что значение переменной age неизвестно.

Значение «Значение «undefined»

Оно означает, что «значение не было присвоено».

```
let age;

alert(age); // выведет "undefined"
```

```
let age = 123;

// изменяем значение на undefined
age = undefined;

alert(age); // "undefined"
```

Объекты и символы

- Тип object (объект) – особенный.
- Все остальные типы называются «примитивными», потому что их значениями могут быть только простые значения (будь то строка, или число, или что-то ещё). В объектах же хранят коллекции данных или более сложные структуры.
- Тип symbol (символ) используется для создания уникальных идентификаторов в объектах.

Оператор Оператор typeof

- Вызов typeof x возвращает строку с именем типа:

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof 10n // "bigint"

typeof true // "boolean"

typeof "foo" // "string"

typeof Symbol("id") // "symbol"

typeof Math // "object"  (1)

typeof null // "object"  (2)

typeof alert // "function"  (3)
```

Взаимодействие: alert, prompt, confirm

- alert
- Она показывает сообщение и ждёт, пока пользователь нажмёт кнопку «ОК».

```
alert("Hello");
```

prompt

- Функция prompt принимает два аргумента:

```
result = prompt(title, [default]);
```

title
Текст для отображения в окне.
default
Необязательный второй параметр, который устанавливает начальное значение в поле для текста в окне.

- Этот код отобразит модальное окно с текстом, полем для ввода текста и кнопками OK/Отмена.

```
let age = prompt('Сколько тебе лет?', 100);

alert(`Тебе ${age} лет!`); // Тебе 100 лет!
```

confirm

```
result = confirm(question);
```

- Функция confirm отображает модальное окно с текстом вопроса question и двумя кнопками: OK и Отмена.
- Результат – true, если нажата кнопка OK. В других случаях – false.

```
let isBoss = confirm("Ты здесь главный?");

alert( isBoss ); // true, если нажата OK
```

Преобразование типов

- Чаще всего операторы и функции автоматически приводят переданные им значения к нужному типу.
- Например, alert автоматически преобразует любое значение к строке. Математические операторы преобразуют значения к числам.
- Есть также случаи, когда нам нужно явно преобразовать значение в ожидаемый тип.

Строковое преобразование

```
let value = true;
alert(typeof value); // boolean

value = String(value); // теперь value это строка "true"
alert(typeof value); // string
```

Численное

```
alert( "6" / "2" ); // 3, строки преобразуются в числа
```

```
let str = "123";
alert(typeof str); // string

let num = Number(str); // становится числом 123

alert(typeof num); // number
```

```
let age = Number("Любая строка вместо числа");

alert(age); // NaN, преобразование не удалось
```

Логическое преобразование

- Происходит в логических операциях (позже мы познакомимся с условными проверками и подобными конструкциями), но также может быть выполнено явно с помощью функции Boolean(value).
- Правило преобразования:

1. Значения, которые интуитивно «пустые», вроде 0, пустой строки, null, undefined и NaN, становятся false.
2. Все остальные значения становятся true.

```
alert( Boolean(1) ); // true
alert( Boolean(0) ); // false

alert( Boolean("Привет!") ); // true
alert( Boolean("") ); // false
```

```
alert( Boolean("0") ); // true
alert( Boolean(" ") ); // пробел это тоже true (любая непустая строка это true)
```

Выражения

Любая комбинация переменных и операций, которая может быть вычислена интерпретатором для получения значения, называется **выражением**.

Примеры

alpha + 19

(alpha - 37)*beta/2

Результатом выполнения всех операций, входящих в состав выражения, является значение.

Выражения и операторы - это не одно и то же. **Операторы** являются указанием совершить какое-либо действие и завершаются точкой с запятой. **Выражения** же определяют некоторую совокупность вычислений.

В одном операторе могут присутствовать несколько выражений.

Операнд – то, к чему применяется оператор. Например, в умножении 5 * 2 есть два операнда: левый операнд равен 5, а правый операнд равен 2. Иногда их называют «аргументами» вместо «операндов».

Термины: «унарный», «бинарный», «операнд»

- Унарным называется оператор, который применяется к одному операнду. Например, оператор унарный минус "-" меняет знак числа на противоположный:

```
let x = 1;

x = -x;
alert( x ); // -1, применили унарный минус
```

- Бинарным называется оператор, который применяется к двум операндам. Тот же минус существует и в бинарной форме:

```
let x = 1, y = 3;
alert( y - x ); // 2, бинарный минус вычитает значения
```

Операции

Операция представляет собой символ, благодаря которому производятся некоторые виды вычислений, сравнений или присваиваний с участием одного или нескольких значений.

Типы операций:

арифметические,

присваивания,

сравнения,

логические,

поразрядные (побитовые).

Значения, расположенные по сторонам операции, называются операндами.

Присваивание

Операция присваивания выглядит как знак равенства =, она присваивает значение, стоящее с правой стороны от нее, переменной, стоящей с левой стороны.

Пример:

let x = 20;

let y = x + 32;

```
let a, b, c;

a = b = c = 2 + 2;

alert( a ); // 4
alert( b ); // 4
alert( c ); // 4
```

```
let n = 2;
n += 5; // теперь n = 7 (работает как n = n + 5)
n *= 2; // теперь n = 14 (работает как n = n * 2)

alert( n ); // 14
```

Арифметические операции

Операция	Знак	Ее функция
Сложение	+	Сложение двух значений
Вычитание	-	Вычитание одного из другого
Умножение	*	Перемножение двух значений
Деление	/	Деление одного значения на другое
Получение остатка от деления	%	Деление одного значения на другое и возвращение остатка (деление по модулю)
Инкремент	++	Сокращенная запись добавления 1 к числу
Декремент	--	Сокращенная запись вычитания 1 из числа
Унарное отрицание	-	Превращение положительного числа в отрицательное или отрицательного в положительное

```
alert( 5 % 2 ); // 1
alert( 8 % 3 ); // 2
```

```
alert( 2 ** 2 ); // 4
alert( 2 ** 3 ); // 8
alert( 2 ** 4 ); // 16
```

```
alert(2 + 2 + '1' ); // будет "41"
```

```
let s = "моя" + "строка";
alert(s); // моястрока
```

```
alert( '1' + 2 ); // "12"
alert( 2 + '1' ); // "21"
```

```
alert( 6 - '2' ); // 4, '2' приводится к числу
alert( '6' / '2' ); // 3, оба операнда приводятся к числам
```

Приведение к числу, унарный +

```
// Не влияет на числа
let x = 1;
alert( +x ); // 1

let y = -2;
alert( +y ); // -2

// Преобразует не числа в числа
alert( +true ); // 1
alert( +"" );   // 0
```

- На самом деле это то же самое, что и Number(...), толь

```
let apples = "2";
let oranges = "3";

alert( apples + oranges ); // "23"
```

```
let apples = "2";
let oranges = "3";

// оба операнда предварительно преобразованы в числа
alert( +apples + +oranges ); // 5

// более длинный вариант
// alert( Number(apples) + Number(oranges) ); // 5
```

Инкремент и декремент

Инкремент ++ увеличивает переменную на 1:

```
let counter = 2;
counter++;        // работает как counter = counter + 1, просто запись короче
alert( counter ); // 3
```

Декремент -- уменьшает переменную на 1:

```
let counter = 2;
counter--;        // работает как counter = counter - 1, просто запись короче
alert( counter ); // 1
```

Знак операции инкремента бывает:

- ✔ в **префиксной** форме, когда он расположен перед своим операндом, ++counter.
- ✔ в **постфиксной** форме, когда операнд записан перед знаком ++. counter++.

Префиксная форма возвращает новое значение, в то время как постфиксная форма возвращает старое (до увеличения/уменьшения числа).

```
let counter = 1;
let a = ++counter; // (*)

alert(a); // 2
```

В строке (*) префиксная форма ++counter увеличивает counter и возвращает новое значение 2. Так что alert покажет 2.

Операции сравнения

Операция	Символ	Функция
Равенство	==	Возвращает true, если операнды по обе стороны от операции равны друг другу
Неравенство	!=	Возвращает true, если операнды по обе стороны от операции не равны друг другу
Больше	>	Возвращает true, если операнд слева от операции больше, чем операнд справа от операции
Меньше	<	Возвращает true, если операнд слева от операции меньше, чем операнд справа от операции
Больше или равно	>=	Возвращает true, если левый операнд больше или равен правому операнду
Меньше или равно	<=	Возвращает true, если левый операнд меньше или равен правому операнду
Строгое равенство	===	Возвращает true, если оба операнда равны и относятся к одному типу
Строгое неравенство	!==	Возвращает true, если операнды не равны или не относятся к одному типу

Логические операции

Логические операции позволяют сравнивать результаты работы двух условных операндов с целью определения факта возвращения одним из них или обоими значения true и выбора соответствующего продолжения выполнения сценария.

Операция	Символ	Функция
Логическое И	&&	Возвращает true, если операдны с обеих сторон от операции вернули значение true
Логическое ИЛИ	||	Возвращает true, если операнд с любой из сторон операции вернул true
Логическое НЕ	!	Операция логического НЕ является унарной. Действие операции ! заключается в том, что она меняет значение своего операнда на противоположное

Оператор if (Условное ветвление)

Оператор **if** позволяет интерпретатору JavaScript выполнять те или иные действия в зависимости от условия.

В операторе **if** сначала вычисляется выражение. Если полученный результат условия равен **true** или может быть преобразован в **true**, то оператор, расположенный в теле **if**, выполняется.

Если результат условия равен **false** или преобразуется в **false**, то оператор не выполнится. **Круглые скобки вокруг выражения являются обязательной частью синтаксиса оператора if.**

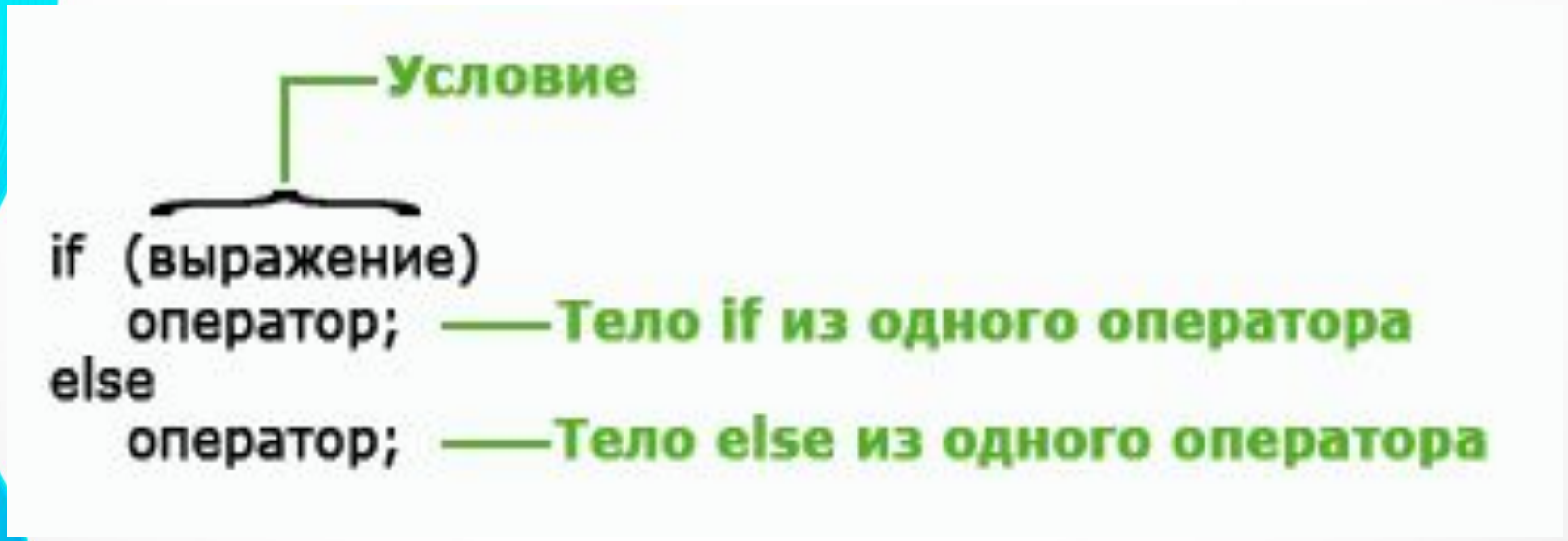
```
let year = prompt('В каком году была опубликована спецификация ECMAScript-2015?', ‘’);
if (year == 2015) alert( 'Вы правы!' );
```

Если мы хотим выполнить более одной инструкции, то нужно заключить блок кода в фигурные скобки:

```
if (year == 2015) {
  alert( "Правильно!" );
  alert( "Вы такой умный!" );
}
```

Оператор if else

Используется, если необходимо совершить одно действие в случае выполнения условия и другое действие в случае невыполнения этого условия.



Конструкция if-else-if

Используется при необходимости проверки несколько условий и выборе правильного.

```
if(условие){
  оператор;
} else if(условие){
  оператор;
} else if(условие){
  оператор;
}
else
  оператор;
```

Иногда, нужно проверить несколько вариантов условия. Для этого используется блок else if.

```
let year = prompt('В каком году была опубликована спецификация ECMAScript-2015?', ‘’);
if (year < 2015) {
   alert( 'Это слишком рано...' );}
else if (year > 2015){
   alert( 'Это поздновато' ); }
else {
   alert( 'Верно!' );
```

Условный оператор «?»

```
if (a < b)
  x = a;
else
  x = b;
```

Тернарный оператор

Тернарный оператор – это оператор, использующий более двух операндов.

С помощью условного оператора предыдущий код можно записать следующим образом:

```
x = (a < b) ? a : b;
```

```
let result = условие ? значение1 : значение2;
```

Сначала вычисляется условие: если оно истинно, тогда возвращается значение1, в противном случае – значение2.

Пример

```
let accessAllowed;
let age = prompt('Сколько вам лет?', '');

if (age > 18) {
  accessAllowed = true;
} else {
  accessAllowed = false;
}

alert(accessAllowed);
```

```
let accessAllowed = (age > 18) ? true : false;
```

Несколько операторов „?“

- Последовательность операторов вопросительного знака ? позволяет вернуть значение, которое зависит от более чем одного условия.

```
let age = prompt('Возраст?', 18);

let message = (age < 3) ? 'Здравствуй, малыш!' :
  (age < 18) ? 'Привет!' :
  (age < 100) ? 'Здравствуйте!' :
  'Какой необычный возраст!';

alert( message );
```

Оператор switch

Конструкция switch заменяет собой сразу несколько if.

Она представляет собой более наглядный способ сравнить выражение сразу с несколькими вариантами.

Конструкция switch имеет один или более блок case и необязательный блок default.

```
switch(x) {
  case 'value1':  // if (x === 'value1')
    ...
    [break]

  case 'value2':  // if (x === 'value2')
    ...
    [break]

  default:
    ...
    [break]
}
```

Пример

```
let a = 2 + 2;

switch (a) {
  case 3:
    alert( 'Маловато' );
    break;
  case 4:
    alert( 'В точку!' );
    break;
  case 5:
    alert( 'Перебор' );
    break;
  default:
    alert( "Нет таких значений" );
}
```

Программа выводит одно из трех сообщений в зависимости от того, какое из чисел находится в переменной x.

Оператор break

Завершает выполнение ветвления **switch**. Управление в этом случае передается первому оператору, следующему за конструкцией switch.

Если значение переменной в операторе switch не совпадает ни с одним из значений констант, указанных внутри ветвления, то управление будет передано в конец switch без выполнения каких-либо других действий.

Пример (без использования оператора break)

```
let a = 2 + 2;

switch (a) {
  case 3:
    alert( 'Маловато' );
  case 4:
    alert( 'В точку!' );
  case 5:
    alert( 'Перебор' );
  default:
    alert( "Нет таких значений" );
}
```

Циклы

Действие циклов заключается в последовательном повторении определенной части вашей программы некоторое количество раз.

Повторение продолжается до тех пор, пока выполняется соответствующее условие.

Когда значение выражения, задающего условие, становится ложным, выполнение цикла прекращается, а управление передается оператору, следующему непосредственно за циклом.

Виды циклов:

- ✔for,
- ✔while;
- ✔do while.

Цикл for

Цикл for организует выполнение фрагмента программы фиксированное число раз. Как правило (хотя и не всегда), этот тип цикла используется, когда известно заранее, сколько раз должно повториться исполнение кода.

```
for (начало; условие; шаг) {
  // ... тело цикла ...
}
```

```
for (let i = 0; i < 3; i++) { // выведет 0, затем 1, затем 2
  alert(i);
}
```

```
for (let i = 0; i < 3; i++) {
  alert(i); // 0, 1, 2
}
alert(i); // ошибка, нет такой переменной
```

```
let i = 0;

for (i = 0; i < 3; i++) { // используем существующую переменную
  alert(i); // 0, 1, 2
}

alert(i); // 3, переменная доступна, т.к. была объявлена снаружи цикла
```

Инициализирующее выражение
Условие выполнения
Итерация
for (i = 0; i < 15; i++)
Примечание: точка с запятой не нужна
оператор;
Тело цикла из одного оператора
for (i = 0; i < 15; i++)
Примечание: точка с запятой не нужна
{
оператор;
оператор;
оператор;
Тело цикла из нескольких операторов - блок
}
Примечание: точка с запятой не нужна

Инициализирующее выражение - представляет из себя оператор присваивания, задающий первоначальное значение переменной, которая выполняет роль счетчика и управляет циклом. **Условие выполнения** - это логическое выражение, определяющее необходимость повторения цикла.

Итерация - выражение, определяющее величину, на которую должно изменяться значение переменной, управляющей циклом, при каждом повторе цикла.

Выполнение цикла for будет продолжаться до тех пор, пока проверка условия дает истинный результат. Как только эта проверка даст ложный результат, цикл завершится, а выполнение программы будет продолжено с оператора, расположенного за циклом.

Цикл while

Содержит условие выполнения цикла, но не содержит ни инициализирующих, ни инкрементирующих выражений.

В случае невыполнения условия при первой проверке тело цикла вообще не исполнялось.

```
while (condition) {
  // код
  // также называемый "телом цикла"
}
```

Код из тела цикла выполняется, пока условие `condition` истинно.

```
let i = 0;
while (i < 3) { // выводит 0, затем 1, затем 2
  alert( i );
  i++;
}
```

Пример

```
let n = 0;
while(n != 5){
  alert(n + " ");
  n++;  //если из кода убрать эту строку, то цикл будет бесконечным
}
```

```
let i = 3;
while (i) { // когда i будет равно 0, условие станет ложным, и цикл остановится
  alert( i );
  i--;
}
```

Цикл do while

Цикл сначала выполнит тело, а затем проверит условие condition, и пока его значение равно true, он будет выполняться снова и снова.

```
do {
  // тело цикла
} while (condition);
```

```
let i = 0;
do {
  alert( i );
  i++;
} while (i < 3);
```

Операторы break и continue

Оператор **break** производит выход из цикла.

Следующим оператором, исполняемым после **break**, будет являться первый оператор, находящийся вне данного цикла.

Пример

```
for(var i = -10; i <= 10; i++){
  if (i > 0) break;
  alert(i + " ");
}
alert("Готово!");
```

```
let sum = 0;

while (true) {

  let value = +prompt("Введите число", '');

  if (!value) break; // (*)

  sum += value;

}
alert( 'Сумма: ' + sum );
```

Переход к следующей итерации: continue

С помощью оператора **continue** можно организовать преждевременное завершение шага итерации цикла. Оператор **continue** осуществляет принудительный переход к следующему шагу цикла, пропуская любой код, оставшийся невыполненым.

Пример

```
for (let i = 0; i < 10; i++) {

  // если true, пропустить оставшуюся часть тела цикла
  if (i % 2 == 0) continue;

  alert(i); // 1, затем 3, 5, 7, 9
}
```

Домашнее задание 1:

1. Возьмите две переменные с числовыми значениями, например: a = 2 и b = 10 (числа могут быть любые). Напишите код, который выводит на экран одну из строк: если истинно условие (a > b) строку "a больше b", если (a < b) тогда строку "a меньше b", если (a == b) тогда строку "a равно b". Для вывода на экран можете использовать document.write() или alert().

Домашнее задание 2:

1. Что делает следующий цикл for? Каково финальное значение переменной sum?

```
var sum = 0;
for(var i = -100; i <= 100; i++){
  sum += i;
}
```

2. Напишите программу, которая использует цикл for для суммирования чисел от 50 до 100. Затем перепишите программу с использованием цикла while.

3. Напишите программу, которая используя цикл while отображает на экране числа от 10 до 0. Затем перепишите программу с использованием цикла for.

Спасибо за внимание