# 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.

## 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.

## Способы размещения JavaScript кода

1. включение кодов JavaScript между тэгами <SCRIPT> и </SCRIPT>;
2. подключение внешнего файла с кодами JavaScript с помощью тэга <SCRIPT>;
3. использование кодов JavaScript непосредственно в тэгах HTML при задании обработчиков событий.

# 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.

## 2. Подключения внешнего файла с JavaScript

Для подключения внешнего файла с JavaScript-кодами используются те же самые тэги <**SCRIPT**> и </**SCRIPT**>, но в отличие от внутреннего встраивания тэг содержит параметр **SRC**, который задает URL-адрес внешнего файла с фрагментами JavaScript.

<**SCRIPT** **src**="**Myscript.js**" >

</**SCRIPT**>

**js** - стандартное расширение для JavaScript-файлов.

Закрывающий тэг </**SCRIPT**> необходим, хотя данные, которые находятся между этими тэгами игнорируются.

Нельзя совместить в одном тэге-контейнере <**SCRIPT**> и </**SCRIPT**> сразу и внутреннее и внешнее подключение кодов JavaScript.

Теги <**SCRIPT**> могут использоваться в любом разделе HTML-документа и в любом количестве.

## 3. Использование событий

Для HTML-тега назначается событие, при выполнении которого выполняется JavaScript код.

**Пример**

<INPUT TYPE=button VALUE="Проведите по мне“
onMouseOver="window.alert(‘Добрый день’);">

# Правила объявления переменных

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

1. Объявление переменных выполняется с помощью ключевого слова **var**:

**var i**;

**var sum**;

2. Один раз использовав ключевое слово var, можно объявить несколько переменных, перечислив их через запятую:

**var i, sum**;

3. Присвоить какое-либо значение переменной можно с помощью оператора присваивания, который обозначется символом (=)

```
var num, msg;
 num = 10;
msg = "Hello";
```

4. Объявление переменных можно совмещать с их инициализацией.

**Инициализация** – это одновременное выполнение двух действий: объявление переменной и присвоение ей начального значения:

```
var num = 10;
var str = "hello";
var i = 0, j = 1, m = 'a';
```

5. Переменные в JavaScript не имеют типа, ей может быть присвоено значение любого типа и позднее этой же переменной может быть присвоено значение другого типа.

```
var x = 10;
x = "текст";
```

6. Если при объявлении переменной ей не было присвоено никакое значение (не была инициализирована), она будет содержать значение undefined (в переводе с англ. неопределённое), пока ей не будет присвоено другое значение.

## Имена переменных. Идентификаторы

**Идентификатор** – последовательность букв, цифр, а также специальных символов.

Идентификаторы выступают в качестве имен переменных, функций, свойств объекта, и т.д.

При выборе идентификатора необходимо учитывать следующее:

1. Идентификаторы не могут совпадать ни с одним из ключевых (зарезервированных) слов JavaScript. Для интерпретатора они имеют специальное значение, т.к. являются частью синтаксиса языка.

2. Идентификаторы могут состоять из одного и более символов.

3. JavaScript чувствителен к регистру символов, следует учитывать регистр при обращении к переменным, функциям и т.д.

4. Идентификаторы могут начинаться с буквы, символа подчеркивания (_) или знака доллара ($). Далее могут следовать любые буквы, цифры, знаки доллара, или символы подчеркивания.

3. JavaScript чувствителен к регистру символов, следует учитывать регистр при обращении к переменным, функциям и т.д.

4. Идентификаторы могут начинаться с буквы, символа подчеркивания (_) или знака доллара ($). Далее могут следовать любые буквы, цифры, знаки доллара, или символы подчеркивания.

# Примеры допустимых идентификаторов

x

my_var

_myCar2

$cash

## 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 | |

# Инструкции

1. Любая программа состоит из последовательности инструкций. Инструкция является указанием на совершение какого-либо действия. Окончание инструкции обозначается символом "точка с запятой" (;).

```
var myNumber = 12;
```

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.

# Чувствительность к регистру

Язык JavaScript чувствителен к регистру символов.

Например: ключевое слово switch должно быть написано как switch, а не Switch или SWITCH, так же и имена переменных myVar, MYVAR или MyVar - будут считаться, как имена трех различных переменных.

# 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 */
```

# Литералы

**Литерал (константа)** – запись в исходном коде программы, представляющая собой обычное фиксированное значение.

Литералы представляют собой константы, непосредственно включаемые в текст программы, в отличие от прочих данных - констант и переменных, обращение к которым осуществляется посредством ссылок. Литералы не могут быть изменены в тексте программы.

```
var num = 14;
var fish = "Кит";
```

**14** и "**Кит**" - литералы,

**num** и **fish** - переменные

# 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

Для представления чисел в JavaScript используется 64-битный формат, определяемый стандартом IEEE 754. Этот формат способен представлять числа в диапазоне от **5e-324** до **1.7976931348623157e+308**.

Все числа в JavaScript представляются вещественными значениями (с плавающей точкой), т.е. нет различий между целыми и вещественными значениями.

```
var bigNumber = 3.52e5;
```

Любому числовому литералу может предшествовать знак минус (-), делающий число отрицательным.

## Специальные числовые значения

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

Переменная **Infinity** хранит специальное значение обозначающее бесконечность, переменная **NaN** также хранит специальное значение NaN (NaN сокращение от англ. Not a Number - не число).

Значение бесконечности можно получить в результате деления числа на 0:

```
alert(123 / 0); // Infinity
```

# Строки

В JavaScript строка - это неизменяемая, упорядоченная последовательность 16-битных значений, каждое из которых представляет символ Юникода. Строки состоят из нуля и более символов. Символы включают в себя буквы, цифры, знаки пунктуации, специальные символы и пробелы. Строки должны быть заключены в кавычки. Использовать можно одиночные кавычки (апострофы) или двойные кавычки.

```
var myColor = "red";
var myColor = 'red';
```

Строки заключённые в двойные кавычки могут содержать символы одиночных кавычек и наоборот.

"одинарные 'кавычки' внутри двойных"

'здесь "наоборот" '

# 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.

# 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**:

```
var x = 20;
var y = x + 32;
```

# 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 |

# Инкремент и декремент

```
var count = count + 1;
```

или

```
count += 1;
```

или

```
++count;
```

Операция ++ инкрементирует - увеличивает на единицу.

## Знак операции инкремента бывает:

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

```
totalWeight = avg * ++count;
totalWeight = avg *count ++;
```

# Составные операции присваивания

```
var x = 5;
x = x + 30;
```

```
x += 30;
```

| | | | | |
|---|---|---|---|---|
| += | -= | *= | /= | %= |

Составные операции присваивания записываются более кратко, чем их несоставные эквиваленты.

# 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.

Логические операции можно применять при необходимости одновременной проверки более одного условия и использования результатов этой проверки.

| 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 (если)

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.**

Если тело содержит нескольких операторов в одном блоке (тело), то они заключаются в фигурные скобки. Таким образом строки кода рассматриваются как один оператор.

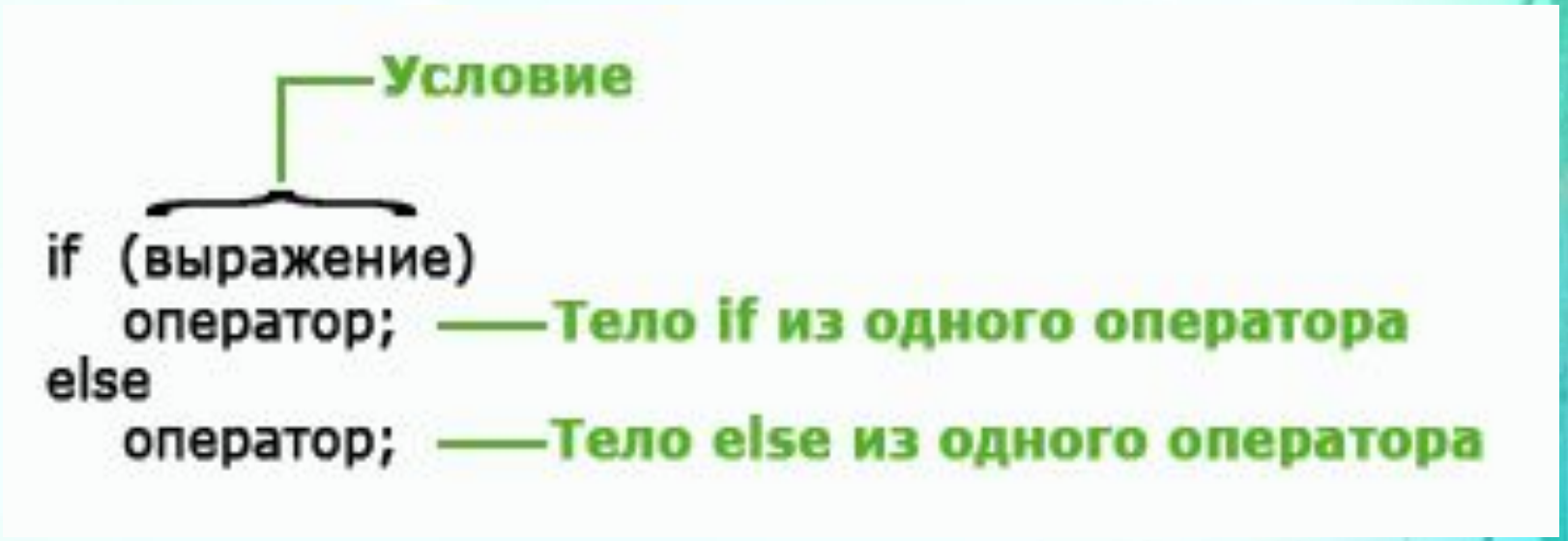
Фигурные скобки рекомендуется использовать всегда, даже когда оператор один. Это улучшает читаемость кода.

**Пример**

```
var num = prompt("Введите любое число","");
 if(num > 5){
 document.write("Число: ", num, " больше 5");
}
```

# Оператор if else

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



Условие

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

Тело if из нескольких операторов

Тело else из нескольких операторов

**Пример**

```
var num = prompt("Введите любое число","");
 if(num >= 10){
  document.write("Число: ", num, " больше или равно 10");
}
else {
  document.write("Число: ", num, " меньше 10");
}
```

# Вложенный оператор if

Оператор if является вложенным, если он находится внутри другого оператора if или else.

```
if(i)
{
  if(a) оператор 1;
  if(b) оператор 2;
  else  оператор 3; //этот else ассоциирован с if(b)
}
else оператор 4; //этот else ассоциирован с if(i)
```

Последний оператор else не относится к if(a) потому, что он находится не во внутреннем блоке, этот else ассоциирован с if(i). Внутренний else ассоциирован с if(b), потому что этот if — ближайший к нему.

# 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;
```

Условные выражения в такой конструкции вычисляются сверху вниз. Как только обнаружится истинное условие, выполняется связанный с ним оператор, а все остальные операторы в многоступенчатой конструкции игнорируются.

Если ни одно из условий не является истинным, то выполняется последний оператор else, который зачастую служит в качестве условия, устанавливаемого по умолчанию. Когда же последний оператор else отсутствует, а все остальные проверки по условию дают ложный результат, то никаких действий вообще не выполняется.

## 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().

## 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;
```

# Пример

```
<html>
<body>
 <script>
  var x;
  var a = 10;
  var b = 15;
  x = (a < b) ? a : b;
  document.write(x);
</script>
</body>
</html>
```

# Оператор switch

Используется, если в программе присутствует большое дерево ветвлений и все ветвления зависят от значения какой-либо одной переменной.

Оператор **switch** сравнивает значение переменной с различными вариантами. При сравнении используется операция строгого равенства "===".

# Пример

```
var x = 3;
switch(x){
case 1:                          //if(x === 1)
    document.write("x равен 1");
    break;
case 2:                          //if(x === 2)
    document.write("x равен 2");
    break;
case 3:                          //if(x === 3)
    document.write("x равен 3");
    break;
}
```

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)

```
var x = 1+1;
switch(x){
case 1:
    document.write("x равен 1");
case 2:
    document.write("x равен 2");
case 3:
    document.write("x равен 3");
}
```

## Ключевое слово default

Предназначено для того, чтобы программа могла выполнить некоторую последовательность действий в том случае, если ни одно из значений констант не совпало со значением переменной в операторе **switch**.

# Пример

```
var x = 3+3;
switch(x){
case 1:
    document.write("x равен 1");
    break;
case 2:
    document.write("x равен 2");
    break;
case 3:
    document.write("x равен 3");
    break;
default:
    document.write("С такими значениями не работаю");
}
```

# 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.

# Пример

На экран выводятся квадраты целых чисел от 0 до 14:

```
var i;
for(i = 0; i < 15; i++){
  document.write("квадрат числа " + i + " = " + (i * i) + "<br>");
}
```


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.

```
while(условие){
  //оператор(ы)
}
```

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

## Пример

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

# Цикл do while

Условие выполнения цикла располагается не перед, а после тела цикла.

**Пример**

```
var n = 0;
do{
  document.write(n + " ");
  n++;
}while(n != 5);
```

Используется ,когда необходимо выполнить тело цикла хотя бы один раз вне зависимости от истинности проверяемого условия.

## 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.

# 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;   //завершить цикл, как только значение
                     //переменной i станет положительным
  document.write(i + " ");
}
document.write("Готово!");
```

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(var i = 0; i <= 100; i++){
  if((i % 2) != 0) continue;    //перейти к следующему шагу итерации
  document.write(i + " ");
}
```