Конспект JavaScript Васильева

Сценарный язык (программа выполняется под управлением ОС, сценарий под управлением браузера).

Перевод программного кода в машинный осуществляется посредством компилятора(переводит и выполняет) или интерпретатора(выполняет когда переводит по-командно). // JavaScript интерпретируемый язык.

IDE - Entegrated Development Environment Интегрированная Среда Разработки.

//(к человеку должно относиться только как к цели, но не как к средству (категорический императив Канта)

//HTML (сокращение от английского HyperText Markup Language)

это дескриптор - <открывающий>(не парный) </закрывающий>(парный).

//сценарий выполняется по мере загрузки документа. Если сценарий находится в заглавном блоке веб-документа, то его выполнение начнется практически сразу при загрузке документа в браузер. Если сценарий находится в блоке основного тела документа, то выполняется он по мере загрузки блока с кодом.

// <дескриптор атрибут=значение>

<script>код</script> внутренний

<script src=“внешний\_файл.js></script> внешний. Если файл со сценарием находится в папке, отличной от той, где находится файл веб-документа, значением атрибута src указывается

полный путь к файлу сценария.

Значения, которыми оперируют в программном коде JavaScript, относятся к одному из следующих типов:

• текстовая строка;

• числовое значение;

• логическое значение (истина или ложь);

• объект;

• функция.

!!! А теперь очень важный момент: в языке JavaScript тип переменной не фиксируется. «;» обязательно использовать (хоть и написано обратное). Объявление переменной var a (не используется «=»). Так же можно объединять: var a=“значение“. Несколько переменных перечисляются через запятую. Переменные должны начинаться с буквы, \_, $. дальше может быть ещё и цифра.

Наряду с переменными в JavaScript используется достаточно много

операторов. Обычно их разделяют на четыре группы:

• арифметические операторы;

• операторы сравнения;

• логические операторы;

• побитовые операторы.

Все операторы, за исключением опера-

торов инкремента и декремента, бинарные — у них по два операнда.

Операторы инкремента и декремента унарные.

+ Оператор сложения. Результатом выражения A+B является сумма

значений числовых операндов A и B

- Оператор вычитания. Результатом выражения A-B является разность

значений числовых операндов A и B

\* Оператор умножения. Результатом выражения A\*B является произведение значений числовых операндов A и B

/ Оператор деления. Результатом выражения A\*B является отношение

значений числовых операндов A и B

% Остаток от целочисленного деления. Результатом выражения A%B

является остаток от деления нацело значения операнда A на значение

операнда B. Операнды могут быть целыми или нецелыми числами

++ Оператор инкремента. В результате вычисления выражения A++

(постфиксная форма) или ++A (префиксная форма) операнд A увеличивает свое значение на 1. Таким образом, инструкция A++, равно как и инструкция ++A, эквивалентна команде A=A+1

-- Оператор декремента. При вычислении выражений --A (префиксная

форма) и A-- (постфиксная форма) значение операнда A уменьшается

на 1. Таким образом, каждая из инструкций --A или A-- эквивалентна

команде A=A-1.

Oперандами оператора + могут быть не только числа, но и текстовые значения. Если оба операнда — текстовые, то выполняется конкатенация строк. Если один из операндов — текст, а другой операнд — число, то выполняется автоматическое приведение числового значения к текстовому с последующей конкатенацией строк.

Помимо этого, допускается использование других арифметических

операторов с текстовыми операндами, которые являются текстовым представлением числа.

Операторы сравнения.

Все операторы бинарные (у каждого по два операнда), а результатом возвращается значение логического типа (true, если соответствующее соотношение истинно, и false в противном случае).Подчеркнем лишь принципиальные отличия между операторами == и ===, а также != и !==. 123="123" является значение true. Значение выражения 123==="123" равно false. Проблема в автоматическом преобразовании типов.

Список операторов:

== Оператор «равно». Результатом выражения A==B является значение

true, если значение операнда A равно значению операнда B. В противном случае значение выражения равно false. Если значения операндов A и B относятся к разным типам, автоматически предпринимается попытка приведения значений к одному типу для проведения сравнения

!= Оператор «неравно». Результатом выражения A!=B является значение true, если значение операнда A не равно значению операнда B. В противном случае значение выражения равно false. Для значений операндов A и B разных типов автоматически выполняется приведение к одному типу (если возможно) для проведения сравнения === Оператор «строго равно». Результатом выражения A===B является

значение true, если значение операнда A равно значению операнда B.

В противном случае значение выражения равно false. В отличие от

оператора == в данном случае приведение к одному типу значений

операндов не выполняется

!== Оператор «строго неравно». Результатом выражения A!==B является значение true, если значение операнда A не равно значению операн да B. В противном случае значение выражения равно false. В отличие от оператора != не выполняется приведение к одному типу значений операндов

> Оператор «больше». Результатом выражения A>B является значение

true, если значение операнда A больше значения операнда B. В противном случае значение выражения равно false

>= Оператор «больше или равно». Результатом выражения A>=B является значение true, если значение операнда A больше или равно значению операнда B. В противном случае значение выражения равно false

< Оператор «меньше». Результатом выражения A<B является значение

true, если значение операнда A меньше значения операнда B. В противном случае значение выражения равно false

<= Оператор «меньше или равно». Результатом выражения A<=B является значение true, если значение операнда A меньше или равно значению операнда B. В противном случае значение выражения равно false.

Логические операторы:

&& Оператор для вычисления логического и. Оператор бинарный. Результатом выражения A&&B при обоих логических операндах A и B является значение true, если оба операнда равны true. Если хотя бы один из двух операндов равен false, результатом выражения A&&B является значение false

|| Оператор для вычисления логического или. Оператор бинарный.

Результатом выражения A||B (при условии, что оба операнда A и B

относятся к логическому типу) является значение true, если хотя бы

один из операндов равен true. Если оба операнда равны false, результатом выражения A||B является значение false

! Оператор логического отрицания. Унарный оператор. Если значение

операнда A равно true, то результат выражения !A равен false. И наоборот, если значение операнда равно false, то результатом выражения !A является значение true.

Если в выражении A&&B **оба** операнда интерпретируются как истинные, результатом возвращается значение **второго** операнда. Если

**хоть один** операнд интерпретируется как ложный, результатом выражения A&&B возвращается значение **первого** «**ложного**» операнда.

• Результатом выражения A||B возвращается первый «истинный»

операнд — если такой есть. Если **оба** операнда интерпретируются

как ложные, то результатом выражения A||B возвращается значение **второго** операнда.

числовые значения при необходимости трансформируются

в логические по следующему правилу: ненулевые числа интерпретируются как логическое значение true, а нулевые числа интерпретируются как логическое значение false. Поэтому, например, значение выражения 1&&2 равно 2, поскольку оба операнда 1 и 2 ненулевые и интерпретируются как true, а результатом возвращается значение второго операнда (реальное значение, а не интерпретируемое). Поэтому получаем результат 2. А вот результатом выражения 2&&0 является

значение 0 — значение первого ложного операнда. Далее, значением

выражения 0&&false является число 0, а значением выражения false&&0 является false. Объяснение простое: если хотя бы один из операндов интерпретируется как ложный, то результатом возвращается значение первого «ложного» операнда. Первым ложным операндом в выражении 0&&false является 0, а первым ложным операндом в выражении false&&0 является false. результатом выражения !1 является значение false, а результатом выражения !0 является значение true.

//Чтобы узнать, как некоторый «объект» A будет интерпретироваться

при преобразовании к логическому типу, можно воспользоваться выражением !!A, в котором использовано двойное логическое отрицание (логическое отрицание логического отрицания).

//логическое исключающее или ^ соответствует выражение !(A&&B)&&(A||B). Его значение равно true при разных значениях операндов A и B (когда один операнд равен true, а другой операнд равен false), и его значение равно false при совпадающих значениях аргументов.

Побитовые или поразрядные операторы:

//вспоминаем правила бинарного сложения чисел. А именно, 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1 и 1 + 1 = 10. 101=1\*2^2+0\*2^1+1\*2^0=4+0+1=5. признаком отрицательности числа является единичный старший бит

& Оператор побитового и. Результат выражения A&B представляет со-

бой число, побитовое представление которого получается на основе

побитовых представлений чисел A и B. Сравниваются биты попарно:

по одному биту из A и из B. При сравнении двух битов результат равен 1, если оба бита равны 1. В противном случае (если хотя бы один из двух битов равен 0) результат сравнения равен 0

| Оператор побитового или. Результат выражения A|B вычисляется на

основе побитовых представлений чисел A и B. Сравниваются соответствующие биты в представлении чисел A и B. При сравнении битов результат равен 1, если хотя бы один бит равен 1. Если оба биты равны 0, результат сравнения равен 0

^ Оператор побитового исключающего или. Результат выражения

A^B представляет собой число, побитовое представление которого

получается на основе побитовых представлений чисел A и B. Сравниваются биты попарно: по одному биту из A и из B. При сравнении двух битов результат равен 1, если один бит равен 1, а другой бит равен 0.

Если оба биты равны 1 или оба биты равны 0, результатом сравнения

битов является число 0

~ Оператор побитового отрицания. Результатом выражения !A является число, получающееся заменой в двоичном представлении числа A

нулей на единицы и единиц на нули (число A при этом не меняется)

<< Оператор побитового сдвига влево. Результатом выражения A<<n является число, которое получается из двоичного представления числа A путем смещения (или сдвига) всего кода влево на n позиций. Старшие биты при этом теряются, а младшие заполняются нулями

>> Оператор побитового сдвига вправо. Результатом выражения A>>n

является число, которое получается из двоичного представления

числа A путем смещения (или сдвига) всего кода вправо на n позиций. Младшие биты при этом теряются, а старшие заполняются значением знакового бита

>>> Оператор побитового сдвига вправо (с замещением старших битов нулями). Оператор побитового сдвига вправо. Результатом выражения

A>>n является число, которое получается из двоичного представления

числа A путем смещения (или сдвига) всего кода вправо на n озиций.

Младшие биты при этом теряются, а старшие заполняются нулями

//Учитывая правила записи и обработки отрицательных чисел, легко

сообразить, что если в переменную A записано некоторое число x, то

результатом выражения ~A является значение –(x + 1). Например, значение выражения ~6 равно -7, а результат выражения ~-5 равен 4.

Если сдвиг влево означает умножение числа на 2, то сдвиг вправо на

одну позицию означает целочисленное деление числа на 2. Поэтому сдвиг кода числа A на n позиций влево означает умножение числа A на 2 в степени n. Это замечание относится как к положительным, так и к отрицательным числам. Оператор >>> отличается от оператора >> тем, что старшие биты заполняются нулями вне зависимости от значения знакового бита.

Таблица 1.12. Сокращенные формы оператора присваивания в JavaScript

Оператор Пример использования Эквивалентная команда

+= A+=B A=A+B

-= A-=B A=A-B

\*= A\*=B A=A\*B

/= A/=B A=A/B

%= A%=B A=A%B

&= A&=B A=A&B

|= A|=B A=A|B

^= A^=B A=A^B

<<= A<<=n A=A<<n

>>= A>>=n A=A>>n

>>>= A>>>=n A=A>>>n

Тернарный оператор

условие ? Значение1 : значение2

true/false ? True : false;

преобразование типов•

!==0 числа могут использоваться в качестве логических значений, они интерпретируются как true, нуль интерпретируется как false.

• Если в арифметическом выражении с вычислением разности, произведения, частного (но не сложения!), побитового сдвига или

другой побитовой операции имеются операнды, являющиеся текстовым представлением числа, такое текстовое представление автоматически преобразуется в число. Например, результатом выражения "300"-"200" является число 100. Или, скажем, результатом выражения "32">>"3" является число 4. Это правило, однако, не распространяется на операцию сложения. Например, результатом выражения "300"+"200" является текстовое значение "300200" (объединение текстовых строк).

• Если один из операндов в выражении вычисления суммы текстовый, а другой операнд числовой, то числовой операнд приводится к текстовому формату и выполняется конкатенация строк. Это же имеет место в том случае, если текстовый операнд является представлением числа. Выше отмечалось, что результат выражения 300+"200" равен "300200", — так же, как и результат выражения "300"+200.

• Для приведения текстового представления числа к числовому типу можно указать унарный оператор плюс перед соответствующим текстовым литералом. Например, результатом выражения +"300"+200 является число 500. Такой же результат получим при вычислении значения выражения +"300"+ +"200". Здесь использованы два оператора плюс подряд (но между ними есть пробел!). Это не оператор инкремента, а два разных оператора. Первый оператор плюс — бинарный, а второй оператор плюс — унарный (знаковый). Данная схема работает, даже если в текстовом литерале «спрятано» отрицательное число. Например, выражение +"300"+ +"-200" имеет смысл. Значение выражения есть число 100. То есть здесь унарный оператор «плюс» выполняет роль «извлекателя» числа из текста.

Приоритетность операторов в JavaScript

Приоритет (по убыванию)

Круглые скобки (используются для

группировки выражений) ( и )

Постфиксная форма инкремента ++

Постфиксная форма декремента --

Логическое отрицание !

Побитовая инверсия ~

Унарный «плюс» +

Унарный «минус» -

Префиксная форма инкремента ++

Префиксная форма декремента --

Умножение \*

Деление /

Остаток от целочисленного деления %

Сложение +

Вычитание -

Побитовый сдвиг влево <<

Побитовый сдвиг вправо >>

Побитовый оператор сдвига вправо >>>

(с заполнением старшего бита нулями)

Меньше <

Меньше или равно <=

Больше >

Больше или равно >=

Равенство ==

Неравенство !=

Строгое равенство ===

Строгое неравенство !==

Побитовое «и» &

Побитовое «исключающее или» ^

Побитовое «или» |

Логическое «и» &&

Логическое «или» ||

Тернарный оператор ?:

Операторы присваивания = += -= \*= /= %= <<= >>= >>>= &= ^= |=

**Управляющие инструкции.**

**Условные операторы**

if(условие){// первый блок команд}

else{//второй блок команд}

Среди выполняемых в условном операторе команд могут быть другие условные операторы. В этом случае говорят о вложенных условных операторах. Вложенные условные операторы позволяют создавать множественные точки ветвления в программном коде.

while(условие){// команды}

Если блок команд состоит из одной команды, фигурные скобки могут отсутствовать.

do{

// команды

}while(условие)

в отличие от while при первом запуске блока команды выполняются до проверки условия

for(первый\_блок;второй\_блок;третий\_блок){

// команды}

Команды в первом блоке выполняются один и только один раз в самом начале выполнения оператора цикла for. Некоторые (или даже все) блоки могут быть пустыми. Пустой второй блок эквивалентен условию true. Формально это означает бесконечный цикл, но на практике в теле оператора цикла можно разместить инструкцию break, которая приведет к завершению выполнения оператора цикла. Причем это касается не только оператора цикла for,но и операторов while и do-while.

switch(условие){

case значение\_1:

// команды 1-ого блока

break

case значение\_2:

// команды второго блока

break

// прочие case-блоки

case значение\_N:

// команды N-ого блока

break

default:

// команды блока по умолчанию

}

**Функции**

Под функцией подразумевают некоторый блок кода(подпрограмму), у которого есть имя и который можно вызвать через данное имя. Что позволяет многократно использовать один и то же код. Помимо имени могут быть аргументы(или параметры), которые передаются ей при вызове. Может возвращать (или не возвращать) результат.

Описание функции — это по большому счету определение того про-

граммного кода, который должен выполняться при вызове функции.

Данный код должен быть «правильно оформлен».

Описываться функция может в любом месте сценария

function название\_функции(аргументы){

// команды

}

Описанную в соответствии с таким шаблоном функцию в програм-

мном коде вызываем командой вида

название\_функции(аргументы)

Описание функции — всего лишь план действий на случай, если

функция будет вызываться. Данное замечание касается и аргументов

функции.

Если функция возвращает значение, то в теле функции используют

инструкцию return, после которой указывается возвращаемое функ-

цией значение. Выполнение инструкции return приводит к заверше-

нию выполнения функции. Если функция не возвращает результат,

то в ней можно использовать инструкцию return, после которой ника-

кое значение не указывается.

Видимость переменных

*Глобальная* переменная — которую можно объявить в одном сценарии, а использовать в другом, при условии что сценарии загружаются в одном документе.

*Локальная* переменная существует только пока выполняется код функции. По завершении функции локальные переменные удаляются из памяти. // в отличии от Java {} не являются областью видимости, она определяется границами функции. Крайне важно использование инструкции var иначе она будет глобальной. // НЕ ВЫПОЛНЯЕТСЯ КОД НЕ ВЫЗВАННОЙ ФУНКЦИИ // В ТЕЛЕ ФУНКЦИИ ЛОКАЛЬНОЙ ПЕРЕМЕННОЙ МОЖНО ПРИСВОИТЬ ЗНАЧЕНИЕ И ТОЛЬКО ПОСЛЕ ЭТОГО ОБЪЯВИТЬ ПЕРЕМЕННУЮ!!!

Локальная имеет приоритет.

Что бы в теле функции одновременно использовать локальную и глобальную переменные надо у глобальной ставить префикс **window.**имя \_переменной.

Аргументы функции.

При передаче аргументов функции передаётся не непосредственно аргумент, а его техническое значение копии (создаваемое автоматически). Это ПЕРЕДАЧА ПО ЗНАЧЕНИЮ.

//////////////////////////////////////////////////////////////////

команды для запоминания:

alert(«some\_text»);

var x="3+(5\*2+6)/4"; document.write(**eval**(x)) выдаст 7

name=prompt(var); //это alert с полем для ввода текста (var), который сохраняется в переменную name.

Примечание: Полная инструкция вызова метода выглядела бы как window.prompt(). Но объект окна window можно не использовать.

var z=9\*Math.random()-2; // случайное от -2 до 7 по-умолчанию диапазон от 0 до 1. Если такое число умножить на 9, получим случайное число в диапазоне возможных значений от 0 до 9. Если от полученного результата отнять 2, получим число в диапазоне возможных значений от –2 до 7.

rnd=6**+**Math.floor(10\*Math.random())// отбрасывает дробную часть, получается 6+0 до 6+9

function f(x){… return smth}

document.write('<img src="function.png" width="100" height="100"> <br>')

document.write("Значение функции: <b>"+**f(z)**+"</b>")вызов функции и интересно обыгрывается <b></b>.

любопытно: for(var n=100,s=0,k=1;k<=n;k++){s+=k\*k}

var txt+=n+"<sup>2</sup> = " // то есть мы можем использовать перменную n *за пределами цикла*.

//формула суммы квадратов натуральных чисел:

12+22+...+n2=(n(n+10)(2n+1))/6

// 6/4=1,5

// **Простыми** называются числа, которые не имеют никаких иных делителей, кроме единицы и самого себя.

**Совершенным** называется число, сумма делителей которого (не учитывая самого числа) равняется данному числу. (6,28)