JavaScript

Тег «script»

Программы на JavaScript могут быть вставлены в любое место HTML-документа с помощью тега <script>.

Для примера:

```
<!DOCTYPE HTML>
<html>

<body>

  <p>Перед скриптом...</p>

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

  <p>...После скрипта.</p>

</body>

</html>
```

Тег <script> содержит JavaScript-код, который автоматически выполнится, когда браузер его обработает.

Внешние скрипты

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

Файл скрипта можно подключить к HTML с помощью атрибута src:

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

Здесь /path/to/script.js – это абсолютный путь до скрипта от корня сайта. Также можно указать относительный путь от текущей страницы. Например, src="script.js" или src="./script.js" будет означать, что файл "script.js" находится в текущей папке.

Можно указать и полный URL-адрес. Например:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lodash.js"></script>
```

Для подключения нескольких скриптов используйте несколько тегов:

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

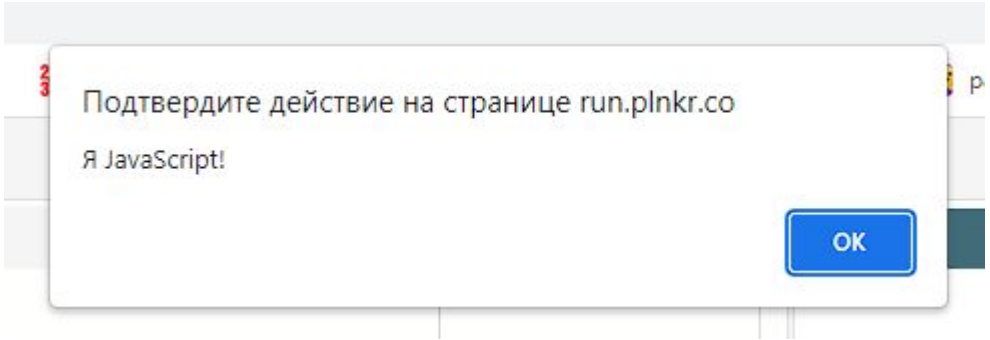
Если атрибут src установлен, содержимое тега script будет игнорироваться.

```
<script src="file.js">
  alert(1); // содержимое игнорируется, так как есть атрибут src
</script>
```

Задачи

1. Создайте страницу, которая отобразит сообщение «Я JavaScript!».

2. Возьмите решение предыдущей задачи и измените его. Извлеките содержимое скрипта во внешний файл alert.js, лежащий в той же папке.



Структура кода

Инструкции

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

Мы уже видели инструкцию alert('Привет, мир!'), которая отображает сообщение «Привет, мир!».

В нашем коде может быть столько инструкций, сколько мы захотим. Инструкции могут отделяться точкой с запятой.

Например, здесь мы разделили сообщение «Привет Мир» на два вызова alert:

```
alert('Привет'); alert('Мир');
```

Обычно каждую инструкцию пишут на новой строке, чтобы код было легче читать:

```
alert('Привет');
alert('Мир');
```

Комментарии

Комментарии могут находиться в любом месте скрипта. Они не влияют на его выполнение, поскольку движок просто игнорирует их.

Однострочные комментарии начинаются с двойной косой черты //.
Пример:

```
// 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');
```

Строгий режим — "use strict"

На протяжении долгого времени JavaScript развивался без проблем с обратной совместимостью. Новые функции добавлялись в язык, в то время как старая функциональность не менялась.

Преимуществом данного подхода было то, что существующий код продолжал работать. А недостатком – что любая ошибка или несовершенное решение, принятое создателями JavaScript, застревали в языке навсегда.

Так было до 2009 года, когда появился ECMAScript 5 (ES5). Он добавил новые возможности в язык и изменил некоторые из существующих. Чтобы устаревший код работал, как и раньше, по умолчанию подобные изменения не применяются. Поэтому нам нужно явно их активировать с помощью специальной директивы: "use strict".

Директива выглядит как строка: "use strict" или 'use strict'. Когда она находится в начале скрипта, весь сценарий работает в «современном» режиме.

Например:

```
"use strict";

// этот код работает в современном режиме
...
```

Проверьте, что "use strict" находится в первой исполняемой строке скрипта, иначе строгий режим может не включиться.

```
alert("some code");
// "use strict" ниже игнорируется - он должен быть в первой строке

"use strict";

// строгий режим не активирован
```

Переменная

Переменная – это «именованное хранилище» для данных. Мы можем использовать переменные для хранения товаров, посетителей и других данных.

Для создания переменной в JavaScript используйте ключевое слово let.

Приведённая ниже инструкция создаёт (другими словами: объявляет или определяет) переменную с именем «messa

```
1  let message;
```

Теперь можно поместить в неё данные, используя оператор присваивания =:

```
let message;

message = 'Hello'; // сохранить строку 'Hello' в переменной с именем message
```

Для краткости можно совместить объявление переменной и запись данных в одну строку:

```
let message = 'Hello!'; // определяем переменную и присваиваем ей значение

alert(message); // Hello!
```

Мы также можем объявить несколько переменных в одной строке:

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

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

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

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

Повторное объявление вызывает ошибку

```
let message = "Это";

// повторение ключевого слова 'let' приводит к ошибке
let message = "Другое"; // SyntaxError: 'message' has already been declared
```

Имена переменных
В JavaScript есть два ограничения, касающиеся имён переменных:

Имя переменной должно содержать только буквы, цифры или символы $ и _.
Первый символ не должен быть цифрой.
Примеры допустимых имён:

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

Если имя содержит несколько слов, обычно используется верблюжья нотация, то есть, слова следуют одно за другим, где каждое следующее слово начинается с заглавной буквы: myVeryLongName.

Самое интересное – знак доллара '$' и подчёркивание '_' также можно использовать в названиях. Это обычные символы, как и буквы, без какого-либо особого значения.

Эти имена являются допустимыми:

```
let $ = 1; // объявили переменную с именем "$"
let _ = 2; // а теперь переменную с именем "_"

alert($ + _); // 3
```

Регистр имеет значение
Переменные с именами apple и APPLE – это две разные переменные.

Константы

Чтобы объявить константную, то есть, неизменяемую переменную, используйте const вместо let:

```
const myBirthday = '18.04.1982';

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

Широко распространена практика использования констант в качестве псевдонимов для трудно запоминаемых значений, которые известны до начала исполнения скрипта.

Названия таких констант пишутся с использованием заглавных букв и подчёркивания.

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";

// ...когда нам нужно выбрать цвет
let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

let и const ведут себя одинаково по отношению к лексическому окружению, области видимости.

Для «var» не существует блочной области видимости
Область видимости переменных var ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока.

```
if (true) {
  var test = true; // используем var вместо let
}

alert(test); // true, переменная существует вне блока if
```

Так как var игнорирует блоки, мы получили глобальную переменную test.

А если бы мы использовали let test вместо var test, тогда переменная была бы видна только внутри if:

```
if (true) {
  let test = true; // используем let
}

alert(test); // Error: test is not defined
```

Аналогично для циклов: var не может быть блочной или локальной внутри цикла:

```
for (var i = 0; i < 10; i++) {
  // ...
}

alert(i); // 10, переменная i доступна вне цикла, т.к. является глобальной переменной
```

Если блок кода находится внутри функции, то var становится локальной переменной в этой функции:

```
function sayHi() {
  if (true) {
    var phrase = "Привет";
  }

  alert(phrase); // срабатывает и выводит "Привет"
}

sayHi();
alert(phrase); // Ошибка: phrase не определена (видна в консоли разработчика)
```

«var» допускает повторное объявление
Если в блоке кода дважды объявить одну и ту же переменную let, будет ошибка:

```
let user;
let user; // SyntaxError: 'user' has already been declared
```

```
var user = "Pete";

var user; // ничего не делает, переменная объявлена раньше
// ...нет ошибки

alert(user); // Pete
```

Если дополнительно присвоить значение, то переменная примет новое значение:

```
var user = "Pete";

var user = "John";

alert(user); // John
```

«var» обрабатываются в начале запуска функции
Объявления переменных var обрабатываются в начале выполнения функции (или запуска скрипта, если переменная является глобальной).

Другими словами, переменные var считаются объявленными с самого начала исполнения функции вне зависимости от того, в каком месте функции реально находятся их объявления (при условии, что они не находятся во вложенной функции).

```
function sayHi() {
  phrase = "Привет";

  alert(phrase);

  var phrase;
}
sayHi();
```

```
function sayHi() {
  var phrase;

  phrase = "Привет";

  alert(phrase);
}
sayHi();
```

```
function sayHi() {
  phrase = "Привет"; // (*)

  if (false) {
    var phrase;
  }

  alert(phrase);
}
sayHi();
```

Это поведение называется «hoisting» (всплытие, поднятие), потому что все объявления переменных var «всплывают» в самый верх функции.

В примере выше if (false) условие никогда не выполнится. Но это никаким образом не препятствует созданию переменной var phrase, которая находится внутри него, поскольку объявления var «всплывают» в начало функции. Т.е. в момент присвоения значения (*) переменная уже существует.

Объявления переменных «всплывают», но присваивания значений – нет.

```
function sayHi() {
  alert(phrase);

  var phrase = "Привет";
}

sayHi();
```

```
function sayHi() {
  var phrase; // объявление переменной срабатывает вначале...

  alert(phrase); // undefined

  phrase = "Привет"; // ...присвоение - в момент, когда исполнится данная строка кода.
}

sayHi();
```

Поскольку все объявления переменных var обрабатываются в начале функции, мы можем ссылаться на них в любом месте. Однако, переменные имеют значение undefined до строки с присвоением значения.

Существует 2 основных отличия var от let/const:

1. Переменные var не имеют блочной области видимости, они ограничены, как минимум, телом функции.
2. Объявления (инициализация) переменных var производится в начале исполнения функции (или скрипта для глобальных переменных).

Типы данных

JavaScript - динамически типизированный

```
// Не будет ошибкой
let message = "hello";
message = 123456;
```

Число

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

Числовой тип данных (number) представляет как целочисленные значения, так и числа с плавающей точкой.
Существует множество операций для чисел, например, умножение *, деление /, сложение +, вычитание - и так далее.
Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: Infinity, -Infinity и NaN.
Infinity представляет собой математическую бесконечность ∞. Это особое значение, которое больше любого числа.
Мы можем получить его в результате деления на ноль:

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

Или задать его явно

```
alert( Infinity ); // Infinity
```

NaN означает вычислительную ошибку. Это результат неправильной или неопределённой математической операции, например:

```
alert( "не число" / 2 ); // NaN, такое деление является ошибкой
```

Любая математическая операция с NaN возвращает NaN

```
alert( NaN + 1 ); // NaN
alert( 3 * NaN ); // NaN
alert( "не число" / 2 - 1 ); // NaN
```

BigInt

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

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}`;
```

Двойные или одинарные кавычки являются «простыми», между ними нет разницы в JavaScript.

Обратные же кавычки имеют расширенную функциональность. Они позволяют нам встраивать выражения в строку, заключая их в ${…}. Например:

```
let name = "Иван";

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

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

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")
```

Специальное значение null не относится ни к одному из типов, описанных выше.
Оно формирует отдельный тип, который содержит только значение null:

```
let age = null;
```

Это просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно».

Значение «undefined»

Оно означает, что «значение не было присвоено».

Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет undefined:

```
let age;

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

Обычно null используется для присвоения переменной «пустого» или «неизвестного» значения, а undefined – для проверок, была ли переменная назначена.

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.

```
var smartphone = {
  "name": "LG G5 se",
  "year": "2016",
  "screen-size": "5.3",
  "screen-resolution": "2560 x 1440",
  "os": "Android 6.0 (Marshmallow)"
};
```

Оператор typeof

```
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)
```

Math — это встроенный объект, который предоставляет математические операции и константы.
Результатом вызова typeof null является "object". Это официально признанная ошибка в typeof, ведущая начало с времён создания JavaScript и сохранённая для совместимости. Конечно, null не является объектом. Это специальное значение с отдельным типом.
Вызов typeof alert возвращает "function", потому что alert является функцией. Функции относятся к объектному типу. Но typeof обрабатывает их особым образом, возвращая "function". Так тоже повелось от создания JavaScript. Формально это неверно, но может быть удобным на практике.

Что выведет этот скрипт?

```
let name = "Ilya";

alert( `hello ${1}` ); // ?

alert( `hello ${"name"}` ); // ?

alert( `hello ${name}` ); // ?
```

Взаимодействие: alert, prompt, confirm

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

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

Это небольшое окно с сообщением называется модальным окном. Понятие модальное означает, что пользователь не может взаимодействовать с интерфейсом остальной части страницы, нажимать на другие кнопки и т.д. до тех пор, пока взаимодействует с окном. В данном случае – пока не будет нажата кнопка «ОК».

Функция **prompt** принимает два аргумента:

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

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

title

Текст для отображения в окне.

default

Необязательный второй параметр, который устанавливает начальное значение в поле для текста в окне.

```
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.

Например:

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

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

Все эти методы являются модальными: останавливают выполнение скриптов и не позволяют пользователю взаимодействовать с остальной частью страницы до тех пор, пока окно не будет закрыто.

На все указанные методы распространяются два ограничения:

1. Расположение окон определяется браузером. Обычно окна находятся в центре.
2. Визуальное отображение окон зависит от браузера, и мы не можем изменить их вид.

Преобразование типов

Строковое преобразование

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

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

Численное преобразование

```
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
```

Правила численного преобразования

Значение	Преобразуется в...
`undefined`	`NaN`
`null`	`0`
`true / false`	`1 / 0`
`string`	Пробельные символы (пробелы, знаки табуляции `\t`, знаки новой строки `\n` и т. п.) по краям обрезаются. Далее, если остаётся пустая строка, то получаем `0`, иначе из непустой строки «считывается» число. При ошибке результат `NaN`.

```
alert( Number("   123   ") ); // 123
alert( Number("123z") );      // NaN (ошибка чтения числа на месте символа "z")
alert( Number(true) );        // 1
alert( Number(false) );       // 0
```

null и undefined ведут себя по-разному. Так, null становится нулём, тогда как undefined приводится к NaN.

Логическое преобразование

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
```

Поддерживаются следующие **математические операторы:**

Сложение +,
Вычитание -,
Умножение *,
Деление /,
Взятие остатка от деления %,
Возведение в степень **.

Сложение строк при помощи бинарного +

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

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

```
alert(2 + 2 + '1' ); // будет "41", а не "221"
```

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

Приведение к числу, унарный +

Унарный, то есть применённый к одному значению, плюс + ничего не делает с числами. Но если операнд не число, унарный плюс преобразует его в число.

```
// 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
```

Присваивание

```
let x = 2 * 2 + 1;

alert( x ); // 5
```

```
let a = 1;
let b = 2;

let c = 3 - (a = b + 1);

alert( a ); // 3
alert( c ); // 0
```

```
let a, b, c;

a = b = c = 2 + 2;

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

Сокращённая арифметика с присваиванием

```
let n = 2;
n = n + 5;
n = n * 2;
```

```
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
```

```
let n = 2;

n *= 3 + 5;

alert( n ); // 16  (сначала выполнится правая часть, выражение идентично n *= 8)
```

Инкремент/декремент

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

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

Операторы ++ и -- могут быть расположены не только после, но и до переменной.
Когда оператор идёт после переменной — это «постфиксная форма»: counter++.
«Префиксная форма» — это когда оператор идёт перед переменной: ++counter.

Поддерживаются следующие **побитовые операторы**:

AND(и) (&)
OR(или) (|)
XOR(побитовое исключающее или) (^)
NOT(не) (~)

Оператор «запятая»

Оператор «запятая» (,) редко применяется и является одним из самых необычных. Иногда он используется для написания более короткого кода, поэтому нам нужно знать его, чтобы понимать, что при этом происходит.

```
let a = (1 + 2, 3 + 4);

alert( a ); // 7 (результат вычисления 3 + 4)
```

Задачи

Простая страница

Создайте страницу, которая спрашивает имя у пользователя и выводит его.

Постфиксная и префиксная формы

Чему будут равны переменные a, b, c и d в примере ниже?

```
let a = 1, b = 1;

let c = ++a; // ?
let d = b++; // ?
```

Результат присваивания

Чему будут равны переменные a и x после исполнения кода в примере ниже?

```
let a = 2;

let x = 1 + (a *= 2);
```

Преобразование типов

Какой результат будет у выражений ниже?

```
"" + 1 + 0
"" - 1 + 0
true + false
6 / "3"
"2" * "3"
4 + 5 + "px"
"$" + 4 + 5
"4" - 2
"4px" - 2
"  -9  " + 5
"  -9  " - 5
null + 1
undefined + 1
" \t \n" - 2
```

Исправьте сложение

Ниже приведён код, который запрашивает у пользователя два числа и показывает их сумму.

Он работает неправильно. Код в примере выводит 12 (для значения полей по умолчанию).

В чём ошибка? Исправьте её. Результат должен быть 3.

```
let a = prompt("Первое число?", 1);
let b = prompt("Второе число?", 2);

alert(a + b); // 12
```

Работа с переменными

1. Создайте переменную num и присвойте ей значение 3. Выведите значение этой переменной на экран с помощью метода alert.

2. Создайте переменные a=10 и b=2. Выведите на экран их сумму, разность, произведение и частное (результат деления).

3. Создайте переменные c=15 и d=2. Просуммируйте их, а результат присвойте переменной result. Выведите на экран значение переменной result.

4. Создайте переменные a=10, b=2 и c=5. Выведите на экран их сумму.

5. Создайте переменные a=17 и b=10. Отнимите от a переменную b и результат присвойте переменной c. Затем создайте переменную d, присвойте ей значение 7. Сложите переменные c и d, а результат запишите в переменную result. Выведите на экран значение переменной result.

Работа со строками

1. Создайте переменную str и присвойте ей значение 'Привет, Мир!'. Выведите значение этой переменной на экран.

2. Создайте переменные str1='Привет, ' и str2='Мир!'. С помощью этих переменных и операции сложения строк выведите на экран фразу 'Привет, Мир!'.

3. Создайте переменную name и присвойте ей ваше имя. Выведите на экран фразу 'Привет, %Имя%!'.

4. Создайте переменную age и присвойте ей ваш возраст. Выведите на экран 'Мне %Возраст% лет!'.

Функция prompt

1. Спросите имя пользователя с помощью методы prompt. Выведите с помощью alert сообщение 'Ваше имя %имя%'.

2. Спросите у пользователя число. Выведите с помощью alert квадрат этого числа.

Обращение к символам строки

1. Создайте переменную str и присвойте ей значение 'abcde'. Обращаясь к отдельным символам этой строки выведите на экран символ 'a', символ 'c', символ 'e'.

2. Создайте переменную num и присвойте ей значение '12345'. Найдите произведение (умножение) цифр этого числа.

Практика

1. Напишите скрипт, который считает количество секунд в часе, в сутках, в месяце.

2. Создайте три переменные - час, минута, секунда. С их помощью выведите текущее время в формате 'час:минута:секунда'.

3. Создайте переменную, присвойте ей число. Возведите это число в квадрат. Выведите его на экран.

Работа с присваиванием и декрементами

1. Переделайте этот код так, чтобы в нем использовались операции +=, -=, *=, /=. Количество строк кода при этом не должно измениться.

```
var num = 47;
num = num + 7;
num = num - 18;
num = num * 10;
num = num / 15;
alert(num);
```

2. Переделайте этот код так, чтобы в нем использовались операции ++ и --. Количество строк кода при этом не должно измениться.

```
var num = 10;
num = num + 1;
num = num + 1;
num = num - 1;
alert(num);
```