Частное учреждение образования

«Колледж бизнеса и права»

|  |
| --- |
| УТВЕРЖДАЮ  Заведующий  методическим кабинетом  \_\_\_\_\_\_\_\_\_\_Е.В.Паскал  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

|  |  |
| --- | --- |
| Специальность:  2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Скриптовые языки программирования» |

**Лабораторная работа № 5**

**Инструкционно-технологическая карта**

Тема: Операторы сравнения, арифметические операторы, присваивание.

Цель: Научиться использовать операторы сравнения, арифметические операторы, присваивание.

Время выполнения: 2 часа

# Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.
2. Реализовать алгоритм решения задачи.
3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
4. Ответить на контрольные вопросы.

# Теоретические сведения

## Операторы сравнения

Многие операторы сравнения известны из математики.

В JavaScript они записываются так:

* Больше/меньше: a > b, a < b.
* Больше/меньше или равно: a >= b, a <= b.
* Равно: a = = b. Обращая внимание, для сравнения используется двойной знак равенства = =. Один знак равенства a = b означал бы присваивание.
* Не равно. В математике обозначается символом ≠, но в JavaScript записывается как a != b.

В этом разделе рассказывается про то, какие бывают сравнения, как язык с ними работает и с какими неожиданностям можно встретиться.

В конце будет хороший рецепт того, как избегать «причуд» сравнения в JavaScript.

#### [Результат сравнения имеет логический тип](https://learn.javascript.ru/comparison" \l "rezultat-sravneniya-imeet-logicheskiy-tip)

Все операторы сравнения возвращают значение логического типа:

* true – означает «да», «верно», «истина».
* false – означает «нет», «неверно», «ложь».

Например:

alert( 2 > 1 ); // true (верно)

alert( 2 == 1 ); // false (неверно)

alert( 2 != 1 ); // true (верно)

Результат сравнения можно присвоить переменной, как и любое значение:

let result = 5 > 4; // результат сравнения присваивается переменной result

alert( result ); // true

#### [Сравнение строк](https://learn.javascript.ru/comparison" \l "sravnenie-strok)

Чтобы определить, что одна строка больше другой, JavaScript использует «алфавитный» или «лексикографический» порядок.

Другими словами, строки сравниваются посимвольно.

Например:

alert( 'Я' > 'А' ); // true

alert( 'Коты' > 'Кода' ); // true

alert( 'Сонный' > 'Сон' ); // true

Алгоритм сравнения двух строк довольно прост:

1. Сначала сравниваются первые символы строк.
2. Если первый символ первой строки больше (меньше), чем первый символ второй, то первая строка больше (меньше) второй. Сравнение завершено.
3. Если первые символы равны, то таким же образом сравниваются уже вторые символы строк.
4. Сравнение продолжается, пока не закончится одна из строк.
5. Если обе строки заканчиваются одновременно, то они равны. Иначе, большей считается более длинная строка.

В примерах выше сравнение **'Я'** > **'А'** завершится на первом шаге, тогда как строки **'Коты'** и **'Кода'** будут сравниваться посимвольно:

1. **К** равна **К**.
2. **о** равна **о**.
3. **т** больше, чем **д**. На этом сравнение заканчивается. Первая строка больше.

#### [Сравнение разных типов](https://learn.javascript.ru/comparison" \l "sravnenie-raznyh-tipov)

При сравнении значений разных типов JavaScript приводит каждое из них к числу.

Например:

alert( '2' > 1 ); // true, строка '2' становится числом 2

alert( '01' == 1 ); // true, строка '01' становится числом 1

Логическое значение true становится 1, а false – 0.

Например:

alert( true == 1 ); // true

alert( false == 0 ); // true

#### [Строгое сравнение](https://learn.javascript.ru/comparison" \l "strogoe-sravnenie)

Использование обычного сравнения **==** может вызывать проблемы. Например, оно не отличает **0** от **false**:

alert( 0 == false ); // true

Та же проблема с пустой строкой:

alert( '' == false ); // true

Это происходит из-за того, что операнды разных типов преобразуются оператором **==** к числу. В итоге, и пустая строка, и **false** становятся нулём.

Как же тогда отличать **0** от **false**?

**Оператор строгого равенства === проверяет равенство без приведения типов.**

Другими словами, если **a** и **b** имеют разные типы, то проверка **a** **===** **b** немедленно возвращает **false** без попытки их преобразования.

Давайте проверим:

alert( 0 === false ); // false, так как сравниваются разные типы

Ещё есть оператор строгого неравенства **!==**, аналогичный **!=**.

Оператор строгого равенства дольше писать, но он делает код более очевидным и оставляет меньше места для ошибок.

#### [Сравнение с null и undefined](https://learn.javascript.ru/comparison" \l "sravnenie-s-null-i-undefined)

Поведение **null** и **undefined** при сравнении с другими значениями – особое:

**При строгом равенстве ===**

Эти значения различны, так как различны их типы.

alert( null === undefined ); // false

При нестрогом равенстве **==**

Эти значения равны друг другу и не равны никаким другим значениям. Это специальное правило языка.

alert(null == undefined); // true

**При использовании математических операторов и других операторов сравнения < > <= >=**

Значения **null/undefined** преобразуются к числам: **null** становится **0**, а **undefined** – **NaN**.

Например, какие забавные вещи случаются, когда применяются эти правила. И, что более важно, как избежать ошибок при их использовании.

#### [Несравненное значение undefined](https://learn.javascript.ru/comparison" \l "nesravnennoe-znachenie-undefined)

Значение **undefined** несравнимо с другими значениями:

alert( undefined > 0 ); // false (1)

alert( undefined < 0 ); // false (2)

alert( undefined == 0 ); // false (3)

Почему же сравнение **undefined** с нулём всегда ложно?

На это есть следующие причины:

Сравнения **(1) и (2)** возвращают **false**, потому что **undefined** преобразуется в **NaN**, а **NaN** – это специальное числовое значение, которое возвращает **false** при любых сравнениях.

Нестрогое равенство **(3)** возвращает **false**, потому что **undefined** равно только **null**, **undefined** и ничему больше.

## [Математика](https://learn.javascript.ru/operators" \l "matematika)

Поддерживаются следующие математические операторы:

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

Первые четыре оператора очевидны, а про **%** и **\*\*** стоит сказать несколько слов.

#### [Взятие остатка %](https://learn.javascript.ru/operators" \l "vzyatie-ostatka)

Оператор взятия остатка %, несмотря на обозначение, никакого отношения к процентам не имеет.

Результат a % b – это остаток от целочисленного деления a на b.

Например:

alert( 5 % 2 ); // 1, остаток от деления 5 на 2

alert( 8 % 3 ); // 2, остаток от деления 8 на 3

#### Возведение в степень \*\*

В выражении **a \*\* b** оператор возведения в степень умножает **a** на само себя **b** раз.

Например:

alert( 2 \*\* 2 ); // 4 (2 умножено на себя 2 раза)

alert( 2 \*\* 3 ); // 8 (2 \* 2 \* 2, 3 раза)

alert( 2 \*\* 4 ); // 16 (2 \* 2 \* 2 \* 2, 4 раза)

Математически, оператор работает и для нецелых чисел. Например, квадратный корень является возведением в степень **1/2**:

alert( 4 \*\* (1/2) ); // 2 (степень 1/2 эквивалентна взятию квадратного корня)

alert( 8 \*\* (1/3) ); // 2 (степень 1/3 эквивалентна взятию кубического корня)

#### Сложение строк при помощи бинарного +

Существуют специальные возможности операторов JavaScript, которые выходят за рамки школьной арифметики.

Обычно при помощи плюса **'+'** складывают числа.

Но если бинарный оператор **'+'** применить к строкам, то он их объединяет в одну:

let s = "моя" + "строка";

alert(s); // моястрока

Внимание, если хотя бы один операнд является строкой, то второй будет также преобразован в строку.

Например:

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

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

Можно увидеть, что не важно, первый или второй операнд является строкой.

Вот пример посложнее:

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

Здесь операторы работают один за другим. Первый **+** складывает два числа и возвращает **4**, затем следующий **+** объединяет результат со строкой, производя действие **4 + '1' = 41**.

Сложение и преобразование строк – это особенность бинарного плюса **+**. Другие арифметические операторы работают только с числами и всегда преобразуют операнды в числа.

Например, вычитание и деление:

alert( 6 - '2' ); // 4, '2' приводится к числу

alert( '6' / '2' ); // 3, оба операнда приводятся к числам

#### Приведение к числу, унарный +

Плюс **+** существует в двух формах: бинарной, которая использовалась выше, и унарной.

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

Например:

// 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(...)**, только короче.

Необходимость преобразовывать строки в числа возникает очень часто. Например, обычно значения полей HTML-формы – это строки. А что, если их нужно, например, сложить?

Бинарный плюс сложит их как строки:

let apples = "2";

let oranges = "3";

alert( apples + oranges ); // "23", так как бинарный плюс объединяет строки

Поэтому здесь используется унарный плюс, чтобы преобразовать к числу:

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

С точки зрения математика, такое изобилие плюсов выглядит странным. Но с точки зрения программиста тут нет ничего особенного: сначала выполнятся унарные плюсы, которые приведут строки к числам, а затем бинарный '**+**' их сложит.

## Присваивание

Отмечая, что в таблице приоритетов также есть оператор присваивания =. У него один из самых низких приоритетов: 3.

Именно поэтому, когда переменной что-либо присваивают, например, x = 2 \* 2 + 1, то сначала выполнится арифметика, а уже затем произойдёт присваивание = с сохранением результата в **x**.

let x = 2 \* 2 + 1;

alert( x ); // 5

Присваивание = возвращает значение

Тот факт, что = является оператором, а не «магической» конструкцией языка, имеет интересные последствия.

Большинство операторов в JavaScript возвращают значение. Для некоторых это очевидно, например сложение + или умножение \*. Но и оператор присваивания не является исключением.

Вызов x = value записывает value в x и возвращает его.

Благодаря этому присваивание можно использовать как часть более сложного выражения:

let a = 1;

let b = 2;

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

alert( a ); // 3

alert( c ); // 0

В примере выше результатом (a = b + 1) будет значение, которое присваивается переменной **a** (то есть 3). Потом оно используется для дальнейших вычислений.

Забавное применение присваивания, не так ли? Нужно понимать, как это работает, потому что иногда это можно увидеть в JavaScript-библиотеках.

Однако писать самим в таком стиле не рекомендуется. Такие трюки не сделают код более понятным или читабельным.

#### Присваивание по цепочке

Существует ещё одна интересная возможность: цепочка присваиваний.

let a, b, c;

a = b = c = 2 + 2;

alert( a ); // 4

alert( b ); // 4

alert( c ); // 4

Такое присваивание работает справа налево. Сначала вычисляется самое правое выражение 2 + 2, и затем результат присваивается переменным слева: c, b и a. В конце у всех переменных будет одно значение.

Опять-таки, чтобы код читался легче, лучше разделять подобные конструкции на несколько строчек:

c = 2 + 2;

b = c;

a = c;

Польза от такого стиля особенно ощущается при быстром просмотре кода.

#### Сокращённая арифметика с присваиванием

Часто нужно применить оператор к переменной и сохранить в ней результат.

Например:

let n = 2;

n = n + 5;

n = n \* 2;

Эту запись можно укоротить при помощи совмещённых операторов += и \*=:

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

Подобные краткие формы записи существуют для всех арифметических и побитовых операторов: /=, -= и так далее.

Вызов с присваиванием имеет в точности такой же приоритет, как обычное присваивание, то есть выполнится после большинства других операций:

let n = 2;

n \*= 3 + 5;

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

Инкремент/декремент

Одной из наиболее частых числовых операций является увеличение или уменьшение на единицу.

Для этого существуют даже специальные операторы:

* 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

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**.

Обе эти инструкции делают одно и то же: увеличивают **counter** на 1.

Есть ли разница между ними? Да, но увидеть её можно, только если будет использоваться значение, которое возвращает ++/--.

Итак, все операторы возвращают значение. Операторы инкремента/декремента не исключение. Префиксная форма возвращает новое значение, в то время как постфиксная форма возвращает старое (до увеличения/уменьшения числа).

Чтобы увидеть разницу, вот небольшой пример:

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.

Пример с постфиксной формой:

let counter = 1;

let a = counter++; // (\*) изменить ++counter на counter++

alert(a); // 1

В строке (\*) постфиксная форма **counter++** также увеличивает **counter**, но возвращает старое значение (которое было до увеличения). Так что **alert** покажет 1.

Подведя итоги:

* Если результат оператора не используется, а нужно только увеличить/уменьшить переменную, тогда без разницы, какую форму использовать:

let counter = 0;

counter++;

++counter;

alert( counter ); // 2, обе строки сделали одно и то же

* Если хочется тут же использовать результат, то нужна префиксная форма:

let counter = 0;

alert( ++counter ); // 1

* Если нужно увеличить и при этом получить значение переменной до увеличения – нужна постфиксная форма:

let counter = 0;

alert( counter++ ); // 0

# Контрольные вопросы

1. Почему значение undefined нельзя сравнить с другими типами?
2. Как произвести возведение в степень, не используя функций?
3. Опишите работу оператора строгого сравнения.
4. В чём разница между операторами «=», «==» и «===»?
5. Как называются операции увеличения и уменьшения значения на единицу? Какие есть два способа описания этих операций и чем они отличаются?

# Задания

**Задание 1.** Выполнить ввод данных, расчёт корней при помощи дискриминанта и вывести результат на экран.

**Задание 2.** Выбрать правильный оператор сравнения между двумя выражениями, чтобы отобразить «**true**»:

var a = "" + 1 + 0;

var b = 9;

alert(a /\*оператор сравнения\*/ b); // true

**Задание 3.** Выбрать правильный оператор сравнения между двумя выражениями, чтобы отобразить «false»:

var a = "" - 1 + 0;

var b = true + false;

alert(a /\*оператор сравнения\*/ b); // false

**Задание 4**. Переделать приведенный код так, чтобы в нем использовались операции +=, -=, \*=, /=, ++, --. Количество строк кода при этом не должно измениться. Код для переделки:

var num = 1;

num = num + 12;

num = num - 14;

num = num \* 5;

num = num / 7;

num = num + 1;

num = num - 1;

alert(num);

**Задание 5**. Написать скрипт, сравнивающий два вводимых пользователем числовых значения.

Результат:

a>b : true

a < b : false

a = b : false

**Задание 6**. Написать скрипт, сравнивающий два вводимых пользователем строковых значения.

Результат:

a>b : true

a < b : false

a = b : false

**Задание 7.** Написать скрипт, выполняющий строгое сравнение двух введённых чисел.

**Задание 8.**

Выполните задание согласно варианта.

|  |  |
| --- | --- |
| №  вар. | Задача |
|  | Тело движется по закону *S =t3 –* 3*t2 +* 2. Вычислить скорость тела в момент времени *t.* Значение *t* ввести с клавиатуры (Функция скорости  есть производная от функции расстояния по времени). |
|  | Найти период  и частоту *T* колебаний в контуре, емкость конденсатора в котором −*С*, индуктивность −*L* Значения *С* и *L* ввести с клавиатуры. Указать единицы измерения вводимых и выводимых величин. |
|  | Написать программу определения катета и площади прямоугольного треугольника, по заданным катету и гипотенузе. Длины катета и гипотенузы вводит пользователь. |
|  | Найти корни квадратного уравнения *a*·*x*2 + *b*·*x* + *c* = 0, введя с экрана коэффициенты *a, b, c* (коэффициент *a* не равен 0), для которых дискриминант положителен. |
|  | Написать программу нахождения гипотенузы и площади прямоугольного треугольника по двум данным катетам. Длины катетов ввести с экрана. |
|  | Вычислить корень уравнения 2*x*/*a* +*b*-12=0 при различных значениях параметров *a, b.* Значения *a, b* ввести с экрана. |
|  | Тело движется по закону .Вычислить скорость тела и расстояние в момент времени *T.* Значение *T* ввести с клавиатуры (функция скорости есть производная от функции расстояния по времени). |
|  | Даны целые положительные числа *a* и *b* (*a* > *b*). На отрезке длины *a* размещено максимально возможное количество отрезков длины *b* (без наложений). Используя операцию деления нацело, найти количество отрезков B, размещенных на отрезке *a.* |
|  | Вычислить *Z = (v1 + v2 + v3)/3*, где *v1,v2,v3* – объемы шаров с радиусами *R1, R2, R3* соответственно. Значения радиусов ввести с экрана. |
|  | Окружность вписана в квадрат заданной площади. Найти площадь квадрата, вписанного в эту окружность. Во сколько раз площадь вписанного квадрата меньше площади заданного? |
|  | Четырехугольник задан координатами своих вершин. Найти его периметр. Вершины ввести с клавиатуры. |
|  | Задан треугольник *АВС* длинами своих сторон *a, b, c.* Определить его медианы (медиана, проведенная к стороне *a,* равна  , значения *a, b, c* ввести с экрана. |
|  | Задан треугольник *АВС* длинами своих сторон *a, b, c* . Определить его биссектрисы (биссектриса, проведенная к стороне *a,* равна  Значения сторон *a, b, c* ввести с экрана. |
|  | Определить периметр треугольника, заданного координатами его вершин. Координаты вершин ввести с клавиатуры. |
|  | Вычислить *Z=(R1 + R2 + R3)/3,* где *R1*, *R2*, *R3* – радиусы шаров с объёмами *V1*, *V2*, *V3* соответственно. Радиус шара вычислять по соответствующей формуле. Значения объемов ввести с экрана. |
|  | Даны длины a, b и c сторон некоторого треугольника. Найти медианы треугольника, сторонами которого являются медианы исходного треугольника. Длина медианы, проведенной к стороне a, равна |
|  | Вычислить рентабельность работы предприятия за месяц по формуле рент = прибыль/себестоимость100%, если себестоимость продукции в текущем месяце уменьшилась по сравнению с прошлым на 5 %. Значение прибыли и себестоимости за прошлый месяц ввести с экрана |
|  | Треугольник задан координатами своих вершин. Найти его площадь, используя формулу Герона Значения a, b и c – длины сторон треугольника. Координаты вершин ввести с клавиатуры. Для вычисления длины отрезка между точками (x1, y1), (x2, y2) использовать формулу |

# Литература

**Диков, А.В.** Клиентские технологии веб программирования: JavaScript и DOM: учебное пособие / А.В. Диков. – СПб: Лань, 2020 – 124 с.

**Читанамбри, Кирупа.** Изучаем React / Кирупа Читанамбри – 2-е изд. – М.: Эксмо, 2019. 368 с.

**Хавербеке, Марейн.** Выразительный JavaScript. Современное веб-программирование. / Марейн Хавербеке – 3-е изд. – СПб.: Питер, 2019. – 480 с.:

Преподаватель Рогалевич А.В.

Рассмотрено на заседании цикловой

комиссии программного обеспечения

информационных технологий №10

Протокол № \_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Председатель ЦК\_\_\_\_\_\_\_\_ В.Ю.Михалевич