02 - Типы данных, операторы и преобразования

1. Типы данных.

В процессе работы компьютерные программы манипулируют значениями, такими как число 3,14 или текст «Hello World». Типы значений, которые могут быть представлены и обработаны в языке программирования, известны как типы данных. **Типы данных** в JavaScript можно разделить на две категории: простые типы и объекты. К категории простых типов в языке JavaScript относятся **числа, строки** и **логические значения**.

Специальные значения **null** и **undefined** являются элементарными значениями, но они не относятся ни к числам, ни к строкам, ни к логическим значениям. Каждое из них определяет только одно значение своего собственного специального типа.

Любое значение в языке JavaScript, не являющееся числом, строкой, логическим значением или специальным значением null или undefined, является **объектом**.

2. Тип данных - Числа.

Число - единый тип, используется как для целых, так и для дробных чисел. Существуют специальные числовые значения **Infinity** (бесконечность) и **NaN** (ошибка вычислений). В отличие от многих языков программирования, в JavaScript не делается различий между целыми и вещественными значениями.

3. Оператор, Операнд, унарный и бинарный оператор.

Для работы с переменными, со значениями, JavaScript поддерживает все стандартные операторы, большинство которых есть и в других языках программирования.

Оператор присваивания - присваивает переменной значения любого типа. Возможно присваивание по цепочке: a = b = c = 2 + 2; Такое присваивание работает справа-налево, то есть сначала вычисляются самое правое выражение 2+2, присваивается в c, затем выполнится b = c и, наконец, a = b.

Операнд – то, к чему применяется оператор. Например: 5 * 2 – оператор умножения с левым и правым операндами.

Унарный оператор - оператор который применяется к одному выражению. Например, оператор унарный минус "-" меняет знак числа на противоположный.

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

Приоритет выполнения. В том случае, если в выражении есть несколько операторов – порядок их выполнения определяется приоритетом. Из школы мы знаем, что умножение в выражении 2 * 2 + 1 выполняется раньше сложения, т.к. его приоритет выше, а скобки явно задают порядок выполнения. Но в JavaScript – гораздо больше операторов, поэтому существует целая **таблица приоритетов.**

Практика: Изучите таблицу приоритетов.

4. Арифметические операции в JavaScript.

Обработка чисел в языке JavaScript выполняется с помощью арифметических операторов. В число таких операторов входят: оператор сложения +, оператор вычитания -, оператор умножения *, оператор деления / и оператор деления по модулю % (возвращает остаток от деления).

Инкремент/декремент: ++, -- Одной из наиболее частых операций в JavaScript, как

и во многих других языках программирования, является увеличение или уменьшение переменной на единицу. Вызывать эти операторы можно не только после, но и перед переменной: i++ (называется «постфиксная форма») или ++i («префиксная форма»). Обе эти формы записи делают одно и то же: увеличивают на 1. Тем не менее, между ними существует разница. Она видна только в том случае, когда мы хотим не только увеличить/уменьшить переменную, но и использовать результат в том же выражении.

Практика: Попробуйте на практике определить разницу между постфиксной и префиксной формой.

Помимо этих простых арифметических операторов JavaScript поддерживает более сложные математические операции, с помощью функций и констант, доступных в виде свойств объекта Math:

```
Math.round(.6) // => 1.0: округление до ближайшего целого
Math.ceil(.6) // => 1.0: округление вверх
Math.floor(.6) // => 0.0: округление вниз
```

Задача: Чему будет равен x в примере ниже?

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

5. Отладка в браузере и console

Перед тем, как двигаться дальше, поговорим об отладке скриптов. Все современные браузеры поддерживают для этого «инструменты разработчика». Исправление ошибок с их помощью намного проще и быстрее. На текущий момент самые многофункциональные инструменты – в браузере Chrome. Также очень хорош Firebug (для Firefox).

6. Двоичное представление вещественных чисел и ошибки округления.

Стандарт представления вещественных чисел, используемый в JavaScript (и практически во всех других современных языках программирования), определяет двоичный формат их представления, который может обеспечить точное представление таких дробных значений, как 1/2, 1/8 и 1/1024. К сожалению, чаще всего мы пользуемся десятичными дробями, такими как 1/10, 1/100 и т.д. Двоичное представление вещественных чисел не способно обеспечить точное представление таких простых чисел, как 0.1.

В будущих версиях JavaScript может появиться поддержка десятичных чисел, лишенная указанных недостатков, связанных с округлением. Но до тех пор предпочтительнее будет использовать масштабируемые целые числа.

```
var x = .3 - .2; // тридцать копеек минус двадцать копеек
var y = .2 - .1; // двадцать копеек минус 10 копеек
x == y // => false: получились два разных значения!
x == .1 // => false: .3-.2 не равно .1
y == .1 // => true: .2-.1 равно .1
```

Практика: Устраните погрешность в примере выше с использованием объекта Math.

7. Тип данных - Строки.

Строка – это неизменяемая, упорядоченная последовательность значений, каждое из которых обычно представляет символ Юникода. Строки в JavaScript являются типом данных, используемым для представления текста. Длина строки – это количество значений, содержащихся в ней. Нумерация символов в строках в языке JavaScript начинается с нуля: первое значение находится в позиции 0, второе – в позиции 1 и т.д.

Пустая строка – это строка, длина которой равна 0.

Одной из встроенных возможностей JavaScript является способность **конкатенировать строки**. Если оператор + применяется к числам, они складываются, а если к строкам – они объединяются, при этом вторая строка добавляется в конец первой.

```
var a = "м о я" + "с т р о к а";
alert( a ); // м о я с т р о к а
```

Вывод: Если хотя бы один аргумент является строкой, то второй будет также преобразован к строке!

Обратите внимание, что, ограничивая строку одинарными кавычками, необходимо проявлять осторожность в обращении с апострофами, употребляемыми в английском языке для обозначения притяжательного падежа и в сокращениях, как, например, в слове «can’t». Поскольку апостроф и одиночная кавычка – это одно и то же, необходимо при помощи символа обратного слэша (\) **«экранировать»** апострофы, расположенные внутри одиночных кавычек.

8. Управляющие последовательности в строковых значениях (литералах)

Символ обратного слэша (\) имеет специальное назначение в JavaScript-строках. Вместе с символами, следующими за ним, он обозначает символ, не представимый внутри строки другими способами. Например, \n – это управляющая последовательность, обозначающая символ перевода строки. Другой пример, упомянутый выше, – это последовательность \', обозначающая символ одинарной кавычки. Эта управляющая последовательность необходима для включения символа одинарной кавычки в строковое значение (литерал), заключенный в одинарные кавычки. Теперь становится понятно, почему мы называем эти последовательности управляющими – здесь символ обратного слэша позволяет управлять интерпретацией символа одинарной кавычки. Вместо того чтобы отмечать ею конец строки, мы используем ее как апостроф:

'You\'re right, it can\'t be a quote'

9. Тип данных - Логические значения.

Логическое значение говорит об истинности или ложности чего-то. Логический тип данных имеет только два допустимых логических значения. Эти два значения представлены литералами **true** и **false.** Логические значения обычно представляют собой результат операций сравнения, выполняемых в JavaScript-программах. Например:

a == 4

Это выражение проверяет, равно ли значение переменной a числу 4. Если да, результатом этого сравнения будет логическое значение true. Если значение переменной a не равно 4, результатом сравнения будет false.

Любое значение в языке JavaScript может быть преобразовано в логическое значение. Следующие значения в результате такого преобразования дают логическое значение false: **undefined, null, 0, -0, NaN, ""** // пустая строка

Все остальные значения, включая все объекты (и массивы), при преобразовании дают в результате значение true. Значение false и шесть значений, которые при преобразовании приводятся к этому значению, называют ложными, а все остальные – истинными. В любом контексте, когда интерпретатор JavaScript ожидает получить логическое значение, ложные значения интерпретируются как false, а истинные значения – как true.

10. Операторы сравнения

Многие операторы сравнения знакомы нам из математики:

- Больше/меньше: a > b, a < b.
- Больше/меньше или равно: a >= b, a <= b.
- Равно a == b. Для сравнения используется два символа равенства '='. Один символ a = b означал бы присваивание.
- «Не равно». В математике он пишется как ≠, в JavaScript – знак равенства с восклицательным знаком перед ним !=.

Сравнение строк. Строки сравниваются побуквенно, сравнение осуществляется как в телефонной книжке или в словаре. Сначала сравниваются первые буквы, потом вторые, и так далее, пока одна не будет больше другой. Иными словами, больше – та строка, которая в телефонной книге была бы на большей странице.

Задача: Объясните, почему выражение вернет true

```
alert( 'а' > 'Я' ); // true
```

Сравнение разных типов. При сравнении значений разных типов, используется числовое **преобразование**. Оно применяется к обоим значениям.

```
alert( '2' > 1 ); // true, сравнивается как 2 > 1
alert( '01' == 1 ); // true, сравнивается как 1 == 1
alert( false == 0 ); // true, false становится числом 0
alert( true == 1 ); // true, так как true становится числом 1.
```

Вывод: Значения разных типов приводятся к числу при сравнении, за исключением строгого равенства === (!==).

Для проверки равенства без преобразования типов используются **операторы строгого равенства** === (тройное равно) и !==.

Сравнение с null и undefined.

- Значения null и undefined равны == друг другу и не равны чему бы то ни было ещё. Это жёсткое правило буквально прописано в спецификации языка.
- При преобразовании в число null становится 0, а undefined становится NaN.

```
alert( null > 0 ); // false
alert( null == 0 ); // false ()
alert(null >= 0); // true
```

Вывод: Любые сравнения с undefined/null, кроме точного ===, следует делать с осторожностью.

11. Преобразование типов

В языке JavaScript значения достаточно свободно могут быть преобразованы из одного типа в другой. Например, если программа ожидает получить строку, а вы передаете ей число, интерпретатор автоматически преобразует число в строку. Если вы укажете не логическое значение там, где ожидается логическое, интерпретатор автоматически выполнит соответствующее преобразование. Свобода преобразований типов значений в JavaScript затрагивает и понятие равенства, и оператор == проверки на равенство выполняет преобразование типов. Переменные в JavaScript не имеют типа: переменной может быть присвоено значение любого типа и позднее этой же переменной может быть присвоено значение другого типа.

Имейте в виду, что возможность преобразования одного значения в другое не означает равенства этих двух значений. Если, например, в логическом контексте используется значение undefined, оно будет преобразовано в значение false. Но это не означает, что undefined == false.

Явные преобразования. Несмотря на то что многие преобразования типов JavaScript выполняет автоматически, иногда может оказаться необходимым выполнить преобразование явно или окажется предпочтительным выполнить явное преобразование, чтобы обеспечить ясность программного кода. Простейший способ выполнить преобразование типа явно заключается в использовании функций **Boolean(), Number(), String() и Object().**

Определенные операторы в языке JavaScript неявно выполняют преобразования и иногда могут использоваться для преобразования типов. Если один из операндов оператора + является строкой, то другой операнд также преобразуется в строку. Унарный оператор + преобразует свой операнд в число. А унарный оператор ! преобразует операнд в логическое значение и инвертирует его. Все это стало причиной появления следующих своеобразных способов преобразования типов, которые можно встретить на практике:

```
x + "" // То же, что и String(x)
+x // То же, что и Number(x). Можно также встретить x-0
!!x // То же, что и Boolean(x). Обратите внимание на два знака !
```

В отличии от Number() **функции parseInt() и parseFloat()** являются более гибкими. Функция parseInt() анализирует только целые числа, тогда как функция parseFloat() позволяет анализировать строки, представляющие и целые, и вещественные числа. Обе функции, parseInt() и parseFloat(), пропускают начальные пробельные символы, пытаются разобрать максимально возможное количество символов числа и игнорируют все, что следует за ними. Если первый непробельный символ строки не является частью допустимого числового значения (литерала), эти
функции возвращают значение NaN.

```
parseInt("3 blind mice") // => 3
parseFloat(" 3.14 meters") // => 3.14
parseInt(".1") // => NaN: целые числа не могут начинаться с "."
parseFloat("$72.47"); // => NaN: числа не могут начинаться с "$"
```

12. Логические операторы

Для операций над логическими значениями в JavaScript есть **|| (ИЛИ), && (И) и ! (НЕ).**

|| (ИЛИ) - Логическое ИЛИ работает следующим образом: "если хотя бы один из аргументов true, то возвращает true, иначе – false".

```
alert( true || true ); // true
alert( false || true ); // true
alert( true || false ); // true
alert( false || false ); // false
```

JavaScript вычисляет несколько ИЛИ слева направо. При этом, чтобы экономить ресурсы, используется так называемый **«короткий цикл вычисления»**. При этом оператор ИЛИ возвращает то значение, на котором остановились вычисления, причем, не преобразованное к логическому типу.

&& (И) - Логическое И возвращает true, если оба аргумента истинны, а иначе – false:

```
alert( true && true ); // true
alert( false && true ); // false
alert( true && false ); // false
alert( false && false ); // false
```

Важно: Приоритет оператора И && больше, чем ИЛИ ||, так что он выполняется раньше.

! (НЕ) - Оператор НЕ. Сначала приводит аргумент к логическому типу true/false. Затем возвращает противоположное значение.

```
alert( !true ); // false
alert( !0 ); // true
```

13. Условные инструкции

Условные инструкции позволяют пропустить или выполнить другие инструкции в зависимости от значения указанного выражения. Эти инструкции являются точками принятия решений в программе, и иногда их также называют инструкциями «ветвления». Если представить, что программа – это дорога, а интерпретатор JavaScript – путешественник, идущий по ней, то условные инструкции можно представить как перекрестки, где программный код разветвляется на две или более дорог, и на таких перекрестках интерпретатор должен выбирать, по какой дороге двигаться дальше. В подразделах ниже описывается основная условная инструкция языка JavaScript –

инструкция if/else, а также более сложная инструкция switch, позволяющая создавать множество ответвлений.

14. Инструкция if

Инструкция if – это базовая управляющая инструкция, позволяющая интерпретатору JavaScript принимать решения или, точнее, выполнять инструкции в зависимости от условий. Инструкция имеет две формы. Первая:

```
if (выражение) {
      инструкция
}
```

В этой форме сначала вычисляется выражение. Если полученный результат является истинным, то инструкция выполняется. Если выражение возвращает ложное значение, то инструкция не выполняется. Например:

```
if (username == null) // Если переменная username равна null или undefined
username = "John Doe"; // определить ее
Аналогично:
// Если переменная username равна null, undefined, 0, "" или NaN,
// присвоить ей новое значение.
if (!username) username = "John Doe";
```

Обратите внимание, что скобки вокруг условного выражения являются обязательной частью синтаксиса инструкции if. Синтаксис языка JavaScript позволяет вставить только одну инструкцию после инструкции if и выражения в круглых скобках, однако одиночную инструкцию всегда можно заменить блоком инструкций. Поэтому инструкция if может выглядеть так:

```
if (!address) {
      address = "";
      message = "Пожалуйста, укажите почтовый адрес.";
}
```

Вторая форма инструкции if вводит конструкцию else, выполняемую в тех случаях, когда выражение возвращает ложное значение. Ее синтаксис:

```
if (выражение)
      инструкция1
else
      инструкция2
```

Эта форма инструкции выполняет инструкцию1, если выражение возвращает истинное значение, и инструкцию2, если выражение возвращает ложное значение. Например:

```
if (n == 1)
      console.log("Получено 1 новое сообщение.");
else
```

```
        console.log("Получено " + n + " новых сообщений.");
```

При наличии вложенных инструкций if с блоками else требуется некоторая осторожность – необходимо гарантировать, что else относится к соответствующей ей инструкции if. Взгляните на следующие строки:

```
i = j = 1;
k = 2;
if (i == j)
      if (j == k)
            console.log("i равно k");
else
      console.log("i не равно j"); // НЕПРАВИЛЬНО!!
```

В этом примере внутренняя инструкция if является единственной инструкцией, вложенной во внешнюю инструкцию if. К сожалению, неясно (если исключить подсказку, которую дают отступы), к какой инструкции if относится блок else. А отступы в этом примере выставлены неправильно, потому что в действительности интерпретатор JavaScript интерпретирует предыдущий пример так:

```
if (i == j) {
      if (j == k)
            console.log("i равно k");
      else
            console.log("i не равно j"); // Вот как!
}
```

Согласно правилам JavaScript (и большинства других языков программирования), конструкция else является частью ближайшей к ней инструкции if. Чтобы сделать этот пример менее двусмысленным и более легким для чтения, понимания, сопровождения и отладки, надо поставить фигурные скобки:

```
if (i == j) {
      if (j == k) {
            console.log("i равно k");
      }
} else { // Вот какая разница возникает из-за добавления фигурных скобок!
      console.log("i не равно j");
}
```

Хотя этот стиль и не используется в данной книге, тем не менее многие программисты заключают тела инструкций if и else (а также других составных инструкций, таких как циклы while) в фигурные скобки, даже когда тело состоит только из одной инструкции. Последовательное применение этого правила поможет избежать неприятностей, подобных только что описанной.

15. Инструкция else if

Инструкция if/else вычисляет значение выражения и выполняет тот или иной фрагмент программного кода, в зависимости от результата. Но что если требуется выполнить один из многих фрагментов? Возможный способ сделать это состоит в

применении инструкции else if. Формально она не является самостоятельной инструкцией JavaScript; это лишь распространенный стиль программирования, заключающийся в применении повторяющихся инструкций if/else:

```
if (n == 1) {
     // Выполнить блок 1
} else if (n == 2) {
     // Выполнить блок 2
} else if (n == 3) {
     // Выполнить блок 3
} else {
     //   Если   ни   одна   из   предыдущих
инструкций else не была выполнена, выполнить
блок 4
}
```

В этом фрагменте нет ничего особенного. Это просто последовательность инструкций if, где каждая инструкция if является частью конструкции else предыдущей инструкции.

16. Инструкция switch

Инструкция if создает ветвление в потоке выполнения программы, а многопозиционное ветвление можно реализовать посредством нескольких инструкций else if. Однако это не всегда наилучшее решение, особенно если все ветви зависят от значения одного и того же выражения. В этом случае расточительно повторно вычислять значение одного и того же выражения в нескольких инструкциях if. Инструкция switch предназначена именно для таких ситуаций. За ключевым словом switch следует выражение в скобках и блок кода в фигурных скобках:

```
switch(выражение) {
     инструкции
}
```

Однако полный синтаксис инструкции switch более сложен, чем показано здесь. Различные места в блоке помечены ключевым словом case, за которым следует выражение и символ двоеточия. Ключевое слово case напоминает инструкцию с меткой за исключением того, что оно связывает инструкцию с выражением, а не с именем. Когда выполняется инструкция switch, она вычисляет значение выражения, а затем ищет метку case, соответствующую этому значению (соответствие определяется с помощью оператора идентичности ===). Если метка найдена, выполняется блок кода, начиная с первой инструкции, следующей за меткой case. Если метка case с соответствующим значением не найдена, выполнение начинается с первой инструкции, следующей за специальной меткой default:. Если метка default: отсутствует, блок инструкции switch пропускается целиком. Работу инструкции switch сложно объяснить на словах, гораздо понятнее выглядит объяснение на примере. Следующая инструкция switch эквивалентна повторяющимся инструкциям if/else, показанным в предыдущем разделе:

```
switch(n) {
case 1: // Выполняется, если n === 1
     // Выполнить блок 1.
     break; // Здесь остановиться
```

```
    case 2: // Выполняется, если n === 2
        // Выполнить блок 2.
        break; // Здесь остановиться
    case 3: // Выполняется, если n === 3
        // Выполнить блок 3.
        break; // Здесь остановиться
    default: // Если все остальное не подходит...
        // Выполнить блок 4.
        break; // Здесь остановиться
}
```

Обратите внимание на ключевое слово break в конце каждого блока case. Инструкция break, описываемая далее в этой главе, приводит к передаче управления в конец инструкции switch и продолжению выполнения инструкций, следующих далее. Конструкции case в инструкции switch задают только начальную точку выполняемого программного кода, но не задают никаких конечных точек. В случае отсутствия инструкций break инструкция switch начнет выполнение блока кода с меткой case, соответствующей значению выражения, и продолжит выполнение инструкций до тех пор, пока не дойдет до конца блока. В редких случаях это полезно для написания программного кода, который переходит от одной метки case к следующей, но в 99% случаев следует аккуратно завершать каждый блок case инструкцией break. (При использовании switch внутри функции вместо break можно использовать инструкцию return. Обе эти инструкции служат для завершения работы инструкции switch и предотвращения перехода к следующей метке case.) Ниже приводится более практичный пример использования инструкции switch; он преобразует значение в строку способом, зависящим от типа значения:

```
function convert(x) {
    switch(typeof x) {
    case 'number': // Преобразовать число в шестнадцатеричное целое
        return x.toString(16);
    case 'string': // Вернуть строку, заключенную в кавычки
        return '"' + x + '"';
    default: // Любой другой тип преобразуется обычным способом
        return x.toString()
    }
}
```

Обратите внимание, что в двух предыдущих примерах за ключевыми словами case следовали числа или строковые литералы. Именно так инструкция switch чаще всего используется на практике, но стандарт ECMAScript позволяет указывать после case произвольные выражения. Инструкция switch сначала вычисляет выражение после ключевого слова switch, а затем выражения case в том порядке, в котором они указаны, пока не будет найдено совпадающее значение с помощью оператора идентичности ===, а не с помощью оператора равенства ==, поэтому выражения должны совпадать без какого-либо преобразования типов.

Задача: Что выведет код ниже?

```
alert( null || 2 || undefined );
alert( alert(1) || 2 || alert(3) );
alert( 1 && null && 2 );
alert( alert(1) && alert(2) );
alert( null || 2 && 3 || 4 );
```

***Конвертер валют**. Разработать приложение, которое организует диалог с пользователем и позволяет вычислить по указанной сумме и курсу евро сумму в гривнях.*

***Стоимость разговора.** Пользователь указывает цену одной минуты исходящего звонка с одного мобильного оператора другому, а также, продолжительность разговора. Необходимо вычислить денежную сумму на которую был произведен звонок.*

***Перевод в дюймы.** Известно, что один дюйм равен двум целым и пятидесяти четырем сотым сантиметра. Разработать программу, с помощью которой возможно организовать перевод указанного пользователем количества сантиметров в дюймы.*

***Простой калькулятор**. Написать программу, которая предлагает пользователю выбрать одну из четырех простейших арифметических операций(+ - * /) и, в зависимости от сделанного выбора, предлагает ввести 2 аргумента и производит соответствующие вычисления (простейший одношаговый калькулятор).*

***Проверка кратности.** Написать программу, которая предлагает пользователю ввести число и определяет кратно ли оно 3, 5, 7.*

***Максимум двух чисел**. Написать программу, которая вычисляет максимум двух введенных пользователем чисел.*

***День недели**. Написать программу, которая предлагает пользователю ввести номер дня недели и в ответ показывает название этого дня.*

***Миллионер** Написать игру "Кто хочет стать миллионером?" в двух вариантах:*
1. Классический - игра идет до первого неправильного ответа. Выиграть можно - лишь ответив на все вопросы правильно.
2. Подсчет очков - за каждый правильный ответ начисляется 5 очков, за каждый неправильный - снимается 10 очков. Счет не может быть отрицательным. Например - игрок ответил на все вопросы, кроме последнего неправильно - результат 5 очков.

***Пенсия**. Пользователь последовательно вводит пол и возраст человека. Программа выводит на экран, пора ему на пенсию или нет. Женщинам после 55, мужчинам после 65.*

***Температура тела**. Пользователь вводит температуру. Ему показывают сообщение о состоянии организма. Больше 41 или меньше 35 - труп, меньше 36.6 - упадок сил, больше 37 - болен, в промежутке 36,6 - 37 - здоров.*

***Площадь фигуры**. Пользователь вводит тип фигуры (треугольник или прямоугольник) и ее размеры. Программа считает площадь и периметр фигуры.*

***Расчет скидки**. Пользователь вводит количество товара и стоимость за штуку. Определить сумму скидки, если при суммарной стоимости 100 гр скидка составляет 3%, 200 – 5%, 300 и более – 7%. Предусмотреть некорректный ввод*

данных - можно вводить только положительные числа, кол-во товара не может быть дробным числом.

***Три числа**. Пользователь вводит 3 числа и тип поиска (минимальное или максимальное). Программа выводит результат.*