Разработка мобильных приложений

Лекция 4

Типы данных в JavaScript

Переменная в JavaScript может содержать любые данные

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

1. Число

- Числовой тип данных (number) представляет как целочисленные значения, так и числа с плавающей точкой.
- Существует множество операций для чисел, например, умножение *, деление /, сложение +, вычитание –
- Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: **Infinity**, **-Infinity** и **NaN**

Infinity

- Infinity представляет собой математическую бесконечность ∞. Это особое значение, которое больше любого числа.
- Его можно получить в результате деления на ноль:
- **alert(1 / 0);** // Infinity
- Или задать явно:
- **alert(Infinity);** // Infinity

NaN (Not a Number)

- NaN означает вычислительную ошибку. Это результат неправильной или неопределённой математической операции, например:
- **alert("не число" / 2);** // NaN
- такое деление является ошибкой
- Любая операция с NaN возвращает NaN:
- **alert("не число" / 2 + 5);** // NaN
- Если где-то в математическом выражении есть NaN, то результатом вычислений с его участием будет NaN.

- Специальные числовые значения относятся к типу «число». Однако, это не числа в привычном понятии этого слова.

2. Строка

- Строка (string) в JavaScript должна быть заключена в кавычки.
- В JavaScript существует три типа кавычек.
- Двойные кавычки: **"Привет"**
- Одинарные кавычки: **'Привет'**
- Обратные кавычки: **`Привет`**

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

- Обратные кавычки же имеют «расширенный функционал». Они позволяют встраивать выражения в строку, заключая их в ${…}.

```
let name = "Иван";
// Вставка переменной
alert( `Привет, ${name}!` );
// Привет, Иван!
// Вставка выражения
alert( `результат: ${1 + 2}` );
// результат: 3
```

3. Булевый (логический) тип

- Булевый тип (boolean) может принимать только два значения: true (истина) и false (ложь).

Пример:

- let isFieldChecked = true;

- Булевые значения также могут быть результатом сравнений:

- let isGreater = 4 > 1;
- alert(isGreater); // true

4. Значение «null»

- Специальное значение null формирует отдельный тип, который содержит только значение null
- let age = null;

- В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках.

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

5. Значение «undefined»

- Специальное значение undefined формирует отдельный тип, который содержит только значение undefined
- Оно означает, что «значение не было присвоено»
- Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет undefined

Пример:

- let x;
- alert(x); // выведет "undefined”

- Технически можно присвоить значение undefined любой переменной:
- let x = 123;
- x = undefined;
- alert(x); // "undefined"

6. Объекты (object)

- Все остальные типы называются «примитивными», потому что их значениями могут быть только простые значения (будь то строка или число, или что-то ещё). Объекты (object) же используются для хранения коллекций данных или более сложных объектов.

7. Символ (symbol)

- Тип symbol (символ) используется для создания уникальных идентификаторов объектов.

Оператор typeof

- Оператор typeof возвращает тип аргумента. Это полезно, когда необходимо обрабатывать значения различных типов по-разному или просто сделать проверку.
- У него есть два синтаксиса:

- typeof x.
- typeof(x).

Вызов typeof x возвращает строку с именем типа

- typeof undefined // "undefined"
- typeof 0 // "number"
- typeof true // "boolean"
- typeof "foo" // "string"
- typeof Symbol("id") // "symbol"
- typeof Math // "object"
- typeof null // "object"
- typeof alert // "function"

- Math – это встроенный объект, который предоставляет математические операции и константы.
- Результатом вызова typeof null является "object". Это неверно. Это официально признанная ошибка в typeof, сохранённая для совместимости.

Преобразование типов

- Чаще всего операторы и функции автоматически приводят переданные им значения к нужному типу.

Строковое преобразование

- Строковое преобразование происходит, когда требуется представление чего-либо в виде строки.
- let value = true;
- alert(typeof value); // boolean
- value = String(value); // теперь value это строка "true"
- alert(typeof value); // string

Числовое преобразование

- Численное преобразование происходит в математических функциях и выражениях.
- 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

- Если строка не может быть явно приведена к числу, то результатом преобразования будет NaN.

- let age = Number("Любая строка вместо числа");
- alert(age); // NaN, преобразование не удалось

Правила численного преобразования:

undefined	NaN
null	0
true / false	1 / 0
string	Пробельные символы по краям обрезаются. Далее, если остаётся пустая строка, то 0, иначе из непустой строки «считывается» число. При ошибке результат NaN.

- alert(Number(" 123 ")); 123
- alert(Number("123z")); NaN
- alert(Number(true)); 1
- alert(Number(false)); 0
- alert(Number(null)); 0
- alert(Number(undefined)); NaN

Сложение «+» объединяет строки

- Почти все математические операторы выполняют численное преобразование. Исключение составляет +. Если одно из слагаемых является строкой, тогда и все остальные приводятся к строкам.
- Тогда они конкатенируются (присоединяются) друг к другу:

- alert(1 + '2'); // '12' (строка справа)
- alert('1' + 2); // '12' (строка слева)

Логическое преобразование

- Значения, которые интуитивно «пустые», вроде 0, пустой строки, null, undefined и NaN, становятся false.
- Все остальные значения становятся true.

- alert(Boolean(1)); true
- alert(Boolean(0)); false

- alert(Boolean("Привет!")); true
- alert(Boolean("")); false

- alert(Boolean("0"); true

Операторы

Операнд

- то, к чему применяется оператор. Например, в умножении 5 * 2 есть два операнда: левый операнд равен 5, а правый операнд равен 2. Иногда их называют «аргументами»

Унарный

- оператор, который применяется к одному операнду. Например, "-" меняет знак числа на противоположный:

- let x = 1;
- x = -x;
- alert(x); // -1, применили унарный минус

Бинарный

- оператор, который применяется к двум операндам. Например:
- let x = 1, y = 3;
- alert(y - x); // 2, бинарный минус

Сложение строк, бинарный +

- alert('1' + 2); // "12"
- alert(2 + '1'); // "21"
- alert(2 + 2 + '1'); // будет "41", а не "221"

Преобразование к числу, унарный плюс +

- если операнд не число, унарный плюс преобразует его в число.
- // не влияет на числа
- let y = -2;
- alert(+y); // -2
- // Преобразует нечисла в числа
- alert(+true); // 1
- alert(+""); // 0

Приоритет операторов

Приоритет	Название	Обозначение
…	…	…
16	унарный плюс	+
16	унарный минус	-
14	умножение	*
14	деление	/
13	сложение	+
13	вычитание	-
…	…	…
3	присваивание	=
…	…	…

Присваивание

- Сначала выполнится арифметика, а уже затем произойдёт присваивание =

- let x = 2 * 2 + 1;
- alert(x); // 5

Присваивание по цепочке

- Присваивание выполняется справа налево
- let a, b, c;

- a = b = c = 2 + 2;

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

Оператор "=" возвращает значение

- Вызов x = value записывает value в x и возвращает его.
- let a = 1;
- let b = 2;
- let c = 3 - (a = b + 1);
- alert(a); // 3
- alert(c); // 0

Остаток от деления %

- alert(5 % 2); // 1, остаток от деления 5 на 2
- alert(8 % 3); // 2, остаток от деления 8 на 3
- alert(6 % 3); // 0, остаток от деления 6 на 3

Возведение в степень **

- Для натурального числа b результат a ** b равен a, умноженному на само себя b раз.
- alert(2 ** 2); // 4 (2 * 2)
- alert(2 ** 4); // 16 (2 * 2 * 2 * 2)
- alert(8 ** (1/3)); // 2 (степень 1/3 эквивалентна взятию кубического корня)

Инкремент/декремент

- Инкремент ++ увеличивает на 1:
- let counter = 2;
- counter++; // работает как counter = counter + 1
- alert(counter); // 3

- Декремент -- уменьшает на 1:
- let counter = 2;
- counter--; // работает как counter = counter - 1
- alert(counter); // 1

Преинкремент/постинкремент

- 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 – постфиксная форма:
- let counter = 0;
- alert(counter++); // 0

Побитовые операторы

- AND(и) (&)
- OR(или) (|)
- XOR(побитовое исключающее или) (^)
- NOT(не) (~)
- LEFT SHIFT(левый сдвиг) (<<)
- RIGHT SHIFT(правый сдвиг) (>>)
- ZERO-FILL RIGHT SHIFT(правый сдвиг с заполнением нулями) (>>>)

Сокращённая арифметика с присваиванием

- Применение оператора к переменной и сохранение результата в ней же.
- 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

Оператор запятая

- Оператор запятая предоставляет нам возможность вычислять несколько выражений, разделяя их запятой ,. Каждое выражение выполняется, но возвращается результат только последнего.
- let a = (1 + 2, 3 + 4);
- alert(a); // 7 (результат 3 + 4)

- // три операции в одной строке
- for (a = 1, b = 3, c = a * b; a < 10; a++) {
- ...
- }

- let a = 1, b = 1;

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

a = 2

b = 2

c = 2

d = 1

- let a = 2;

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

a = 4

x = 5