Базовые конструкции языка
Введение в JavaScript
Объявление переменных
Примитивные типы данных
Приведение типов, оператор присваивания, функции alert, prompt, confirm
Арифметические операции
Условные операторы if и switch
Операторы циклов for, while, do while

Примитивные типы данных

Смотреть материал на видео

Пришло время познакомиться с примитивными типами данных в JavaScript. В самом простом случае, когда мы объявляем переменную вот так:

```
let a = 2;
```

то переменная принимает числовой тип (number). Этот пример показывает, что тип переменной определяется типом данных, которые мы ей присваиваем. Если переменной присвоить строку:

```
let b = "Строка";
```

то переменная b принимает строковый тип. В JavaScript существует семь основных типов данных. Рассмотрим их по порядку.

Числовой тип (number)

Числовой тип данных представляет как целочисленные значения, так и числа с плавающей точкой. Например, обе эти переменные будут относиться к числовому типу:

```
let a = 5;
let b = 7.23;
```

JavaScript «знает» пять основных типов числовых литералов:

- десятичный – целое число (например, 0, 5, -10, -100, 56);
- приближенные к вещественным – приблизительные вещественные числа (например, 6,7, 8.54, -10.34);
- экспоненциальный (научный) – с использованием буквы 'e' (например, 10 = 1e1, 20 = 2e1, 25000 = 25e3, 8700 = 8.7e3);
- восьмеричная: 0o777;
- шестнадцатиричная: 0xff24f.

Однако, на практике чаще всего используются целые и приближенные к вещественным.

Оператор typeof

Чтобы узнать тип переменной используется оператор typeof, который имеет такой синтаксис:

typeof <переменная>;

или

typeof(<переменная>);

его можно вызывать как со скобками, так и без скобок. Обычно используют первый вариант. Выведем тип переменной a:

```
console.log(typeof a);
```

Мы видим значение number – числовой тип.

Infinity и NaN

Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому же 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:

```
let c = 1/0;
```

или задать его явно:

```
let d = Infinity;
let d2 = -Infinity;
```

Мы можем также получить бесконечность, если попытаемся записать в переменную очень большое число, которое выходит за пределы представления чисел в JavaScript. Например, такое:

```
let inf = 1.0e1000;
```

Следующее значение NaN (от *англ.* Not a Number – не число) означает вычислительную ошибку. Это результат неправильной или неопределённой математической операции, например:

```
let c = "строка"/2;
```

Значение NaN «прилипчиво». Любая операция с NaN возвращает NaN:

```
let c = "строка"/2 + 2;
```

Значения Infinity и NaN хотя и относятся к числовым типам, но скорее являются индикаторами, говорящими разработчику скрипта что пошло не так.

Строковый тип (string)

Как мы говорили на предыдущем занятии, строки можно задавать через такие кавычки:

```
let msg1 = "строка 1";
let msg2 = 'строка 2';
let msg3 = `строка 3`;
```

Двойные или одинарные кавычки являются «простыми», между ними нет разницы в JavaScript. Обратные кавычки (апострофы) имеют «расширенный функционал» и были введены в ES6. Они позволяют встраивать выражения в строку, заключая их в ${...}. Например, так:

```
let msg3 = `Значение a = ${a}`;
```

Здесь вместо ${a} будет подставлено значение переменной a и в консоле увидим «Значение a = 5». Вместо переменной можно записать любое выражение языка JavaScript:

```
let msg3 = `Значение a = ${1+2}`;
```

Получим строку «Значение a = 3». Все это работает только с обратными кавычками. Если их заменить на простые, то никакой подстановки не произойдет:

```
let msg3 = 'Значение a = ${a}';
```

Тем, кто знаком с другими языками программирования следует знать, что в JavaScript нет типа данных для отдельного символа (типа char). Здесь есть только строки и даже один символ – это строка из одного символа.

А как быть, если мы хотим в строке записать кавычки? Например, так:

```
let msg1 = "строка "привет"";
```

Есть несколько способов это сделать. Первый – заменить двойные кавычки всей строки на одинарные:

```
let msg1 = 'строка "привет" ';
```

Второй способ – использовать так называемое экранирование символов:

```
let msg1 = "строка \"привет\"";
```

мы здесь перед каждой кавычкой внутри строки поставили обратный слеш. Это называется экранированием символов. На практике используются оба способа, так что выбирайте любой удобный для себя. Кстати, если нужно просто отобразить обратный слеш, то его следует записать так:

```
let msg1 = "строка \\";
```

Булевый (логический) тип

Булевый тип (boolean) может принимать только два значения: true (истина) и false (ложь). Такой тип, как правило, используется для хранения значений да/нет: true значит «да, правильно», а false значит «нет, не правильно». Например:

```
let isWin = true, isCheckedField = false;
```

Булевые значения также могут быть получены из результатов сравнения:

```
let isGreater = 4 > 1;
```

и часто применяются в условных операторах, о которых мы будем говорить на последующих занятиях.

Значение «null»

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

```
let idProcess = 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».

Значение «undefined»

Это свой тип данных, который имеет только одно значение undefined, означающее, что значение не было присвоено. Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет undefined:

```
let arg;
```

По идее мы можем присвоить значение undefined любой переменной:

```
let a = undefined;
```

Но так делать не рекомендуется. Если нам нужно отметить, что переменная не содержит данных, то для этого используется значение null, а undefined – только для проверок: была ли переменная инициализирована или по каким-то причинам ей «забыли» присвоить значение.

Тип Symbol

Это новый тип данных (symbol), который впервые появился в спецификации ES6. Он используется для создания уникальных идентификаторов. Например, так:

```
let id = Symbol();
```

И в большей степени он удобен для дальнейшего развития JavaScript. Чтобы можно было добавлять новые идентификаторы, не меняя код, в котором он уже мог использоваться.

Символы создаются конструктором Symbol() и могут дополняться необязательным описанием:

```
let id2 = Symbol("id2");
```

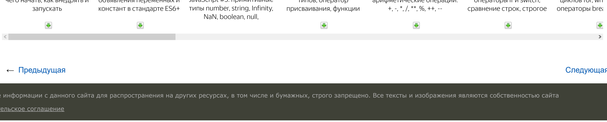
Их уникальность можно проверить оператором идентичности, о котором мы будем говорить на следующих занятиях. Здесь приведу лишь пример:

```
console.log(id === id2);
```

В некоторой степени они похожи на объекты, но относятся к примитивным типам.

Если сейчас вам не совсем понятно, что такое Symbol, просто пропустите этот момент. Когда придет время для его использования, все встанет на свои места.

Видео по теме



JavaScript #1: что это такое, с чего начать, как внедрять и запускать

JavaScript #2: способы объявления переменных и констант в стандарте ES6+

JavaScript #3: примитивные типы number, string, Infinity, NaN, boolean, null,

JavaScript #4: приведение типов, оператор присваивания, функции

JavaScript #5: арифметические операции: +, -, *, /, **, %, ++, --

JavaScript #6: условные операторы if и switch, сравнение строк, строгое

JavaScript #7: циклов for, wh операторы bre

← Предыдущая Следующая →