JavaScript
часть 1

Hello, World!

1. Устанавливаем Node.js

   https://nodejs.org

2. Создаём файл `hello.js` со следующим контентом:

```
let hello = "Hello, World!";
console.log(hello);
```

3. Выполняем в терминале команду `node hello.js`

Комментарии

```
/*
Многострочный
комментарий.
*/

// Однострочный комментарий
```

Объявление и присваивание переменных

```
let x = 42;
x = 12; // OK

let x;
x = 24; // OK

const y = 42;
y = 12; // TypeError: Assignment to constant variable.
```

Типы данных: Number

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

let m = 1 / 0; // Infinity

let k = "строка" * 2 // NaN
```

Тип данных: String

```
let str = "Строки заключаются в двойные кавычки";
str = 'Одинарные кавычки тоже подойдут';
str = `Обратные кавычки для интерполяции`;
```

```
let name = 'Mark';
let greeting = `Hello, ${name}! How are you?`;
// "Hello, Mark! How are you?"
```

Тип данных: Boolean

```
let checked = true;
checked = false;
```

Тип данных: null

```
let age = null;
```

Значение null образует свой отдельный тип, состоящий из единственного значения null.

В JavaScript null не является «нулевым указателем», как в некоторых других языках.
Это специальное значение со смыслом «ничего» или «значение неизвестно».

Тип данных: undefined

```
// переменная объявлена, но в неё ничего не записано
let x;
console.log(x); // выведет "undefined"

let x = 123;
// undefined присвоить можно, но бессмысленно
x = undefined;
```

Значение undefined, как и null, образует свой собственный тип, состоящий из одного этого значения. Оно имеет смысл «значение не присвоено».

Тип данных: Object

```
let user = { name: "Вася" };
```

Коллекции и функции – тоже экзамеляры типа Object.

Оператор typeof

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof true // "boolean"

typeof "foo" // "string"

typeof {} // "object“


typeof null // "object"

typeof function () { } // "function"
```

Математические операторы

```
let x = 1 + 2; // 3, сложение
let y = 5 - 4; // 1, вычитание
let z = 2 * 3; // 6, умножение
let m = 8 / 2; // 4, деление
let n = 10 % 3; // 1, остаток от деления
let p = 2 ** 9; // 512, возведение в степень

// постфиксный инкремент
let a = 3;
let b = a++; // a = 4, b = 3

// префиксный инкремент
let c = 3;
let d = ++c; // c = 4, d = 4

// декремент - аналогично
```

Операторы сравнения

```
let one = 1, two = 2,
    three = 3, four = 4,
    five = 5;

// все выражения принимают значение true

one < two; // меньше
four > three; // больше

one <= one; // меньше или равно
one <= two;

five >= four; // больше или равно
five >= five;

three == three; // равно
three != four; // не равно
```

Сравнение строк

В JavaScript реализовано *лексикографическое* сравнение строк.

```
'Б' > 'А'; // true, сравниваются численные коды Unicode символов

'а' > 'Я'; // true, код у строчной буквы больше, чем у прописной

'Банан' > 'Аят'; // true, сравнение происходит побуквенно

'Вася' > 'Ваня'; // true, т.к. 'с' > 'н'

'Привет' > 'Прив'; // true, т.к. любая буква больше отсутствия буквы
```

Сравнение разных типов

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

```
'2' > 1; // true, сравнивается как 2 > 1

'01' == 1; // true, сравнивается как 1 == 1

false == 0; // true, false становится числом 0

true == 1; // true, так как true становится числом 1
```

Строгое равенство

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

```
// все выражения равны true
0 == false;
'42' == 42;
'' == false;
```

```
// все переменные равны false
0 === false;
'42' === 42;
'' === false;
```

Сравнение с null и undefined

Интуитивно кажется, что null/undefined эквивалентны нулю, но это не так.

Алгоритмы проверки равенства `==` и сравнения `>=` `>` `<` `<=` работают по-разному.

1. Значения `null` и `undefined` равны `==` друг другу и не равны чему бы то ни было ещё.
2. В операторах `>=` `>` `<` `<=` при преобразовании в число `null` становится `0`, а `undefined` становится `NaN`.

```
null > 0; // false
null == 0; // false

null >= 0; // true
```

```
undefined > 0; // false
undefined < 0; // false
undefined == 0; // false
```

Условные операторы

```
if (age < 12) {
   // детство
} else if (age < 16) {
   // отрочество
} else if (age < 25) {
   // юность
} else {
   // молодость
}
```

```
let message = age > 18
   ? 'Welcome'
   : 'Access denied';
```

Если условие не типа Boolean:

- Число `0`, пустая строка `""`, `null` и `undefined`, а также `NaN` преобразуются `false`,
- Остальные значения преобразуются `true`.

Логические операторы

```
let isValidAge = age > 35 && age < 70; // И

let isValidZone = zone == 'RU' || zone == 'BY'; // ИЛИ

let showInvalidAgeError = !isValidAge; // НЕ


// приоритет у && больше, чем у ||
let isPensionAge = sex == 'm' && age > 65 || sex == 'w' && age > 60;


// короткий цикл вычисления
let x = true || neverCalled();
let y = false && neverCalled();
```

Строковое преобразование

Обычно преобразование происходит неявно:

```
let a = true;
console.log(a); // выведет "true"
```

Но можно выполнить и явное преобразование:

```
let b = String(null); // "null"
```

Или почти явное:

```
let c = true + "test"; // "truetest"
let d = "123" + undefined; // "123undefined"
```

Численное преобразование

Явное преобразование:

```
let a = +"123"; // 123
let b = Number("123"); // 123
```

Другие значения:

```
let d = +undefined; // NaN
let c = +null; // 0
let e = +true; // 1
let f = +false; // 0
```

В строках обрезаются все пробельные символы по краям.
Далее, если остаётся пустая строка, то 0, иначе из непустой строки "считывается" число, при ошибке результат NaN

```
let g = +"   \n  123   \n  \n"; // 123
```

```
if ("\n" == true) {
    // код не выполнится
}

if ("\n") {
    // код выполнится
}
```

Логическое преобразование

Для чисел:

```
let a = Boolean(42); // true
let b = Boolean(0); // false
let c = Boolean(NaN); // false
```

Для строк:

```
let d = !!"abc"; // true
let e = !!""; // false
```

Для объектов:

```
let f = {} && true; // true
```

Для специальных значений:

```
if (undefined) { /* код внутри не выполнится */ }
if (null) { /* код внутри не выполнится */ }
```

Циклы

```
let i = 0;
while (i < 3) {
    console.log(i);
    i++;
}
// 0, 1, 2
```

```
let i = 0;
do {
    console.log(i);
    i++;
} while (i < 3);
// 0, 1, 2
```

```
for (let i = 0; i < 3; i++) {
    console.log(i);
}
// 0, 1, 2
```

break, continue, метки — есть

Конструкция switch

```
let arg = prompt("Введите arg:");

switch (arg) {
    case '0':
    case '1':
        alert('Один или ноль');
        break;

    case '2':
        alert('Два');
        break;

    case 3:
        alert('Никогда не выполнится');
        break;

    default:
        alert('Неизвестное значение: ' + arg);
        break;
}
```

Функции

Function Declaration

```
function printGreeting(name) {
    console.log(`Hello, ${name}!`);
}
```

Function Expression

```
let printGreeting = function (name) {
    console.log(`Hello, ${name}!`);
};
```

Arrow Function Expression

```
let printGreeting = name => console.log(`Hello, ${name}!`);
```

Function Declaration

```
function something() {

    printGreeting('Mark Zuckerberg'); // работает

    function printGreeting(name) {
        console.log(`Hello, ${name}!`);
    }

}
```

Function Expression

```
let printGreeting;

if (daypart = 'morning') {
    printGreeting = function() {
        console.log("Go away whoever you are");
    };
} else {
    printGreeting = function (name) {
        console.log(`Hello, ${name}!`);
    };
}

printGreeting();
```

Область видимости переменных

```
console.log(x); // ReferenceError: x is not defined

let x = 42;

// создаём блок фигурными скобками
if (true) {
    let y = 12;
    console.log(x); // 42
}

// объявляем функцию
function foo() {
    console.log(x);
}

console.log(x); // 42
foo(); // 42
console.log(y); // ReferenceError: y is not defined
```