Типы данных в JS

Типы данных

Простые

— Number
— String
— Boolean
— null
— undefined
— Symbol

Составные

— Object
— Array
— Function
— Regexp
— Date
— Error
— Set
— Map

Числа (класс `Number`)

— 8 байт, число с плавающей точкой (-2^{53} .. 2^{53})

— Специальные значения: `+-Infinity`, `NaN`, `-0`

— Арифметические операции: `+ - / * % ** ++ --`

— Битовые операции: `& | ^ ~ << >>`

— Операции сравнения: `< > <= >=`

— Библиотека математических функций `Math`

— Библиотека функций в `Number` и `Number.prototype`

```
56.14; 0xFFED; -45e13;  // литералы чисел
5 + 15;                 // 20
1 === 1.0;              // true
.3 - .2 === .1;         // false
Math.abs(.3-.2-.1) < Number.EPSILON;// true
```

```
Number.isFinite(Infinity);   // false
Number.isInteger(1.1);       // false
Number.isNaN(NaN);           // true
(-12.1).toFixed(3);          // "-12.100"
(77.123).toExponential(3));  // "7.712e+1"
```

Строки (класс `String`)

— Упорядоченная последовательность 16-битных значений

— UTF-16, ограниченная поддержка multibyte chars (code points)

— Операторы: `+` (конкатенация), `[]` (доступ по индексу)

— Свойство `length` - длина строки

```
'I\'m a string!'      // I'm a string!
"I'm a string too!"   // I'm a string too!

"abcd".length;        // 4
"abcd"[2];            // "c"
"abcd".charCodeAt(2); // 99
```

Литералы строк

single quotes / double quotes / backticks (template strings)

```
'my name is "Patches O\'hoolihan"'
"or with double quotes - \"Patches O'hoolihan\""

"Also can use escape sequences: \n \t \\"
"And use utf codes: \uD834\uDF06"          // "And use utf codes: 𝌆"
"Or use code points: \u{1D306}"            // "Or use code points: 𝌆"

`Interpolation is best: ${20 / 4}`         // "Interpolation is best: 5"

`Multiline
 string`         // "Multiline
                 // string"
```

Логические значения (класс `Boolean`)

— Значения `true` и `false`

— "falsy" значения: `false, 0, -0, null, undefined, NaN, ""`

— Операции: `&& || !`

```
true && false;   // false
false || true;   // true
!false;          // true
// Самый простой логический контекст - условие if
if (null) ...    // не выполнится, null - falsy значение
if ("abc") ...   // выполнится

// Операция || возвращает первое не-falsy значение, или последнее из всех
undefined || 5;  // 5
"" || 0;         // 0
// Операция && возвращает первое falsy значение, или последнее из всех
"abcd" && NaN;   // NaN
"abcd" && 1;     // 1

// Операции || и && - "ленивые"
19 || f();       // 19, функция f не вызовется
null && f();     // null, функция f не вызовется
```

`null`, `undefined`

— Два типа данных с единственным значением в каждом

— `null` - отсутствие значения

— `undefined` - значение не определено/не инициализировано

```
let drWatsonName = {
    first:  "John",
    last:   "Watson",
    middle: null      // спойлер: после третьего сезона - "Hamish"!
};

let undef;
undef;     // undefined
```

Объектные обертки над простыми типами

— Простые типы имеют специальные "объектные обертки"

— `Number`, `String`, `Boolean` - функции-конструкторы

— Обертка создается неявно при необходимости

— Явное создание оберток бесполезно и даже вредно

```
let s = "test";

s.trim();           // (new String(s)).trim() - доступ к функции класса String

s.myProp = 4;       // (new String(s)).myProp = 4;
s.myProp;           // (new String(s)).myProp; -> undefined

new Number(5) + new Number(5); // 10, примитивное значение

let result = new Boolean(false);
if (result) {
    // эта ветка выполнится (!)
}
```

Классы встроенных типов данных

— `Number`, `String`, `Boolean` - функции с множеством применений:

— Конструктор класса (имеет смысл для составных типов)

— Приведение типов

— Их свойства - это статические методы/константы класса

— Свойство `prototype` - контейнер для методов класса

```
Number("5678");     // typecast из строки в число
new Number(5678);   // создание объекта класса Number
Number.MAX_VALUE;   // 1.79E+308, статическая константа класса
Number.isFinite(1); // true, статический метод класса

// Функция Number.prototype.toFixed
(5).toFixed(3);     // "5.000", метод класса
```

Приведение типов

— Неявное приведение типов - Очень Плохая Вещь

— "Полуявное" приведение с помощью неявного

— Явное приведение с помощью `Number`, `String`, `Boolean`

```
// Неявное приведение типов. Не повторяйте это ни дома, ни на работе.
5 + "hello";     // "5hello"

// "Полуявное" приведение типов
+ x;     // к числу
x + "";  // к строке
!!x;     // к логическому значению

Number(true);            // 1
Number("15hj");          // NaN
Number.parseInt("15hj"); // 15, "мягче" чем Number("15hj")

String(false);   // "false"
String(15e4);    // "150000"

Boolean(NaN);    // false
Boolean({});     // true, любой объект в логическом контексте дает true
```

Объекты (класс `Object`)

— Неупорядоченная коллекция *свойств*

— Свойство имеет имя и значение

— Имя - строка, значение - любой тип данных

```
let empty = {}; // Пустой объект
let point = {x: 5, y: 6};
let x = 10, y = 15;
let point2 = {x, y};  // то же самое что и {x: x, y: y}

let suffix = "name";
let book = {
    "book title": "A Clockwork Orange",    // Имя свойства с пробелом
    "class": "fiction",                    // Зарезервированное слово
    dynamicId: getId(),                  // Значение свойства - любое выражение
    tags: ["fiction", "Burgess"],        // И любая структура данных
    ["Author's " + suffix]: "Anthony Burgess", // "Author's name"

    getName: function() { ... },  // Метод
    getName2() { ... }            // Метод: сокращенный синтаксис
};
```

Чтение и запись свойств

— Операторы `[]` и `.` - обращение к свойству объекта

— `delete` - удаление свойства

— `in` - существует ли свойство в объекте

— Обращение к несуществующему свойству => `undefined`

```
let obj = {x: 1, "first name": "Shmebulock"},
    suffix = "name";

obj.x;                  // 1
obj["first " + suffix]; // "Shmebulock"
obj["interface"];       // undefined

delete obj.x;
"x" in obj;             // false
"first name" in obj;    // true
```

Полезные методы для работы с объектами

— `Object.keys(obj)` - массив свойств объекта

— `Object.values(obj)` - массив значений свойств

— `Object.entries(obj)` - массив пар "ключ-значение"

— `Object.is()` - проверка эквивалентности, которую мы заслуживаем

— `Object.assign()` - слияние свойств одного объекта в другой

```
obj = {x: 1, y: 2};
Object.keys(obj); // ["x", "y"]

Object.is(NaN, 0/0);        // true
Object.is(null, undefined); // false

const o1 = { a: 1, b: 1, c: 1 },
      o2 = { b: 2, c: 2 };
      o3 = { c: 3 };

const obj = Object.assign({}, o1, o2, o3);
console.log(obj);           // { a: 1, b: 2, c: 3 }
```

Простые значения и ссылки на объекты

— Простые типы иммутабельны и сравниваются по значению

— Объекты изменяемы, сравниваются и передаются по ссылке

```
let s = "hello";
s.toUpperCase();  // -> "HELLO"
s;                // -> "hello"

let o = { x: 1 };
o.x = 2;
o.x;              // -> 2, изменения сохранились

let a = [1, 2, 3];
let b = a;        // b указывает на тот же массив что и a

b[0] = 0, b[3] = 4;
a[0];             // -> 0
a[3];             // -> 4

let c = [0, 2, 3, 4];
a === c;          // -> false, разные ссылки на массивы
```

Деструктурирующее присваивание

— Извлечение значений из объекта в отдельные переменные

— Можно пропускать ненужные значения

— Можно указывать значения по умолчанию

```
let { w, h } = { w: 100, h: 200 };
console.log(w, h);          // -> 100 200

let { w: width, h: height } = { w: 100, h: 200 };
console.log(width, height); // -> 100 200

let w, h;
({ w, h = 300 } = { w: 100 });
console.log(w, h);          // -> 100 300
```

Массивы (класс `Array`)

— Упорядоченная коллекция значений

— Объект с целочисленными именами свойств

— Свойство `length` - длина массива

— `[]` - обращение к элементу массива

```
let a = [1, 2, 3, 4],                  // Создание через литерал
    b = [1, "abc", {x: 5}, [1, 2, 3]], // Может содержать любые типы данных

a.length;          // 4
Array.isArray(a);  // true
Array.isArray({}); // false

2 in a;   // true
4 in a;   // false
a[2];     // 3
a[4];     // undefined
```

Деструктурирующее присваивание (you again!)

```
let [fName, lName] = ["James", "Bond"];
console.log(fName, lName); // "James", "Bond"

let [, lName2] = ["James", "Bond"];
console.log(lName2);       // "Bond"

let [fName = "Mighty", lName = "Anonymous"] = [];
console.log(fName, lName); // "Mighty", "Anonymous"

[a, b] = [b, a];           // Обмен значениями в одну строчку
```

Rest синтаксис

— преобразование список значений => массив

— Rest-переменная всегда должна быть последней в списке

```
let [a, ...b] = [1, 2, 3];
console.log(a, b);          // => 1, [2, 3];

function f(a, ...etc) {
    console.log(a, etc);    // => 1, [2, 3]
}
f(1, 2, 3);

function f(a, ...[b, c]) { // Serious business!
    console.log(a, b, c);    // => 1, 2, 3
}
f(1, 2, 3);
```

Spread синтаксис

— преобразование итератор => список значений

```
// удобное слияние массивов
let colors = ["red", "green"],
    newColors = ["yellow", ...colors, "blue"];

// создание "клона" массива
let colors = [ "red", "green", "blue" ],
    [ ...clonedColors ] = colors;

// Вместо Math.max(arr[0], arr[1], ...);
let arr = [1, 2, 3, 4, 5];
Math.max(...arr);

// Преобразование итератора в массив
let myArray = [1, "abc", true];
let entries = [...myArray.entries()];
// [[0, 1], [1, "abc"], [2, true]];
```

Множества (Класс `Set`)

— Упорядоченная коллекция значений без повторов

— Конструктор `new Set([iterable])`

— Значения сравниваются по правилам `Object.is()` (кроме +-0)

```
let s = new Set();
s.add(5); s.add("5"); s.add(5);
s.size;     // 2, дубликат проигнорирован
s.has("5"); // true
s.clear();
s.size;     // 0

let s2 = new Set(["a", "b", "c"]);
for (const val of s2) console.log(val); // "a", "b", "c"
```

Словари (Класс Map)

— Упорядоченная коллекция пар "ключ-значение"

— Конструктор `new Map([iterable])`

```
let map = new Map([["name", "Nicholas"], ["age", 25]]);

map.has("name");    // true
map.delete("name");
map.has("name"); // false
map.get("name"); // undefined
map.size;        // 1

map.clear();
map.has("age");  // false
map.size;        // 0

map.set("key1", "a");
map.set("key2", "b");
map.forEach(function(value, key) {console.log(`value: ${value}, key: ${key}`)});
// => value: a, key: key1
// => value: b, key: key2
```

Итераторы у классов `Array`, `Set`, `Map`

— `values()` - значения; умолчание для `Array` и `Set`

— `keys()` - ключи; для `Set` то же самое что и `values()`

— `entries()` - пары "ключ-значение"; умолчание для `Map`;

— `Array` => `Map/Set` через конструктор `Map/Set`

— `Map/Set` => `Array` через `Array.from()` или spread: `[...x]`

```
const a = ["a", "b", "c", "a"],
  s  = new Set(a);          // множество с элементами "a", "b", "c"
  s2 = new Set(a.keys());   // множество с элементами 0, 1, 2, 3
  m1 = new Map(a.entries()); // хэш 0 => "a", 1 => "b", 2 => "c", 3 => "a"

  s3 = new Set(m1);         // множество [0, "a"], [1, "b"], [2, "c"], [3, "a"]

  a2 = [...s3.entries()];
  // [[[0, "a"], [0, "a"]], [[1, "b"], [1, "b"]], [[2, "c"], [2, "c"]], [[3, "a"], [3, "a"]]]
```