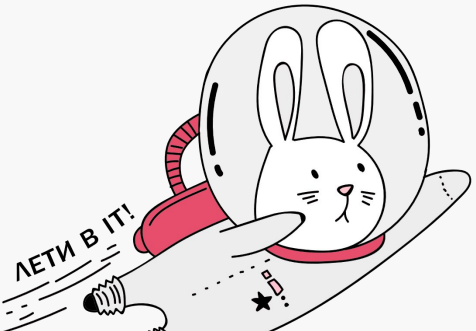
JavaScript

This
Arrays
Objects
Functions



dev {education}

This

В отличие от многих других языков программирования ключевое слово this в javascript не привязывается к объекту, а зависит от контекста вызова. Для упрощения понимания будем рассматривать примеры применительно к браузеру, где глобальным объектом является window. P.S. в Node.js - глобальным объектом является global.

This в JavaScript - это динамическое понятие которое зависит от контекста вызова функции, в котором this описан. Пока функция не вызвана, мы не сможем точно сказать чему будет равен this.

This

```
function foo() {
    this.value = 5;
    console.log(this === window);
}
foo() // true
```

Частным случаем будет случай, в котором прописано выражение “use strict” до декларации нашей функции. В этом случае this будет равен undefined. То есть в консоли выведется false.

```
var myObject = new foo(); // никогда не называйте функции конструкторы с маленькой буквы !!!
console.log(myObject.value === 5); // true
```

При вызове функции с использованием ключевого слова new функция выступает в роли конструктора, и в данном случае this указывает на создаваемый объект.

This

```
var myObject = {
      myself: function() {
            return this;
      }
}

console.log(myObject.myself() === myObject); // true
```

Если функция запускается как свойство объекта, то в this будет ссылка на этот объект. При этом не имеет значения, откуда данная функция появилась в объекте, главное — как она вызывается, а именно какой объект стоит перед вызовом функции.

Arrays

Массив - разновидность объекта (Array - глобальный объект), а также это конечная последовательность упорядоченных элементов любого типа, доступ к каждому элементу в которой осуществляется по его индексу.

Варианты создания массива:

```
var fruits = [element0, element1, ..., elementN];
var fruits = new Array(element0, element1, ..., elementN);
```

var fruits = **new** Array(arrayLength) arrayLength - целое число от 0 до $2^{32} - 1$ (включительно), этот массив будет иметь **пустые** ячейки а не ячейки со значениями **undefined**. Если же аргументом будет другое число произойдёт **RangeError**.

В массиве может храниться ограниченное число элементов любого типа:

```
var arr = [ 99, 'Name', { city: 'Boston' }, true, [1, 2, 3] ];
```

Arrays

Одно из применений массива – это **очередь**.

В классическом программировании так называют упорядоченную коллекцию элементов, такую что элементы добавляются в конец, а обрабатываются – с начала.

Очень близка к очереди еще одна структура данных – **стек**.

Это такая коллекция элементов, в которой новые элементы добавляются в конец и берутся с конца.

Arrays. Methods

pop() - удаляет последний элемент из массива и возвращает его;
push() - добавляет элемент в конец массива;
shift() - удаляет первый элемент из массива и возвращает его;
unshift() - добавляет элемент в начало массива;

```
var values = [5, 8];
values.push(7); values.push(3);
var lastElement = values.pop();
values.unshift(9); values.unshift(2);
var firstElement = values.shift();

console.log(values, firstElement, lastElement);  ===>  [9, 5, 8, 7]   2   3
```

Arrays. Methods

Сравнение методов добавления и удаления:

Методы push/pop выполняются быстро, а shift/unshift – медленно.

Операция shift должна выполнить целых три действия:

1. Удалить нулевой элемент, тем самым освободить ячейку с индексом 0.
2. Произвести смещение всех элементов влево на одну ячейку.
3. Обновить свойство length, уменьшив его на единицу.

У операции unshift похожий алгоритм, только добавление в начало.

Arrays. Methods

of() - создаёт новый экземпляр Array, согласно заданного количества аргументов, несмотря на то, сколько их было передано и какого они типа;
from() - создаёт новый экземпляр Array (мелкую копию) из подобного к массиву или из итерируемого объекта (Set, Map, etc...);
isArray() - выясняет, является ли переданное значение массивом;
un/observe() - **DEPRECATED**. Использовался для асинхронного отслеживания изменений в массивах, похож на **Object.observe()** в объектах. Теперь необходимо использовать глобальный объект **Proxy**;

```
Array.of(1, 2, 3); // [1, 2, 3]
Array.isArray(1, 2, 3); // false
Array.isArray([1, 2, 3]); // true
Array.from(obj, mapFn, thisArg), где thisArg - this для mapFn (коллбек map);
Array.from([1, 2, 3], x => x + x) // [2, 4, 6]
```

Arrays. Methods

find() - возвращает значение первого элемента в массиве, который удовлетворяет условию переданной функции. В противном случае возвращается **undefined**.
findIndex() - возвращает индекс первого элемента в массиве, который удовлетворяет условию переданной функции. В противном случае возвращает **-1**, это означает, что ни один из элементов массива не прошел проверку.

```
var values = [3, 9, 15, -5, 12];
console.log(values.find(element => element > 13)); // 15
console.log(values.findIndex(element => element > 13)); // 2
```

Arrays. Methods

includes() - выясняет, содержит ли массив переданное значение и возвращает соответственно **true** или **false**.
indexOf() - возвращает первый индекс, по которому данный элемент был найден в массиве, в противном случае вернёт **-1**.
lastIndexOf() - возвращает последний индекс, по которому данный элемент был найден в массиве, в противном случае вернёт **-1**.

В методах indexOf и lastIndexOf есть возможность задать значение fromIndex - индекс, с которого начинается поиск. По умолчанию в первом методе значение fromIndex = **0**, а во втором: **arr.length - 1**.

```
var values = [2, 9, 9];
values.includes(2) // true
values.indexOf(9) // 1
values.lastIndexOf(9) // 2

values.includes(7) // false
values.indexOf(7) // -1
values.lastIndexOf(7) // -1
```

Arrays. Methods

forEach() - выполняет переданную функцию один раз для каждого элемента массива.
filter() - создаёт новый массив со всеми элементами, которые удовлетворяют условию переданной функции.
map() - создаёт новый массив с результатами вызова переданной функции на каждом элементе массива, который вызвала эта функция.
reduce() - выполняет функцию reducer (функцию указуем мы) для каждого элемента массива и возвращает единственное значение.
flat() - создаёт новый массив который содержит все элементы вложенных массивов до указанной глубины.

```
var words = ['number', 'string', 'symbol', 'object', 'undefined'];
console.log(words.filter(word => word.length > 6)); // ["undefined"]
```

Arrays. Methods

```
var values = [1, 2, [3, 4, [5, 6]]];
console.log(values.flat()); // [1, 2, 3, 4, [5, 6]]
console.log(values.flat(2)); // [1, 2, 3, 4, 5, 6]

var chars = ['a', 'b', 'c'];
chars.forEach(element => console.log(element)); // a b c

var numbers = [1, 4, 9, 16];
console.log(numbers.map(x => x * 2)); // [2, 8, 18, 32]

var reducer = (accumulator, currentValue) => accumulator + currentValue;
console.log(numbers.reduce(reducer)); // 30
console.log(numbers.reduce(reducer, 5)); // 35
```

Arrays. Methods

every() - проверяет, все ли элементы массива удовлетворяют условию, которое описано в функции, которая передаётся как аргумент и возвращает **true** или **false**.
some() - проверяет, содержит ли массив хотя бы один элемент, для которого выполняется условие в переданной функции как аргумент и возвращает **true** или **false**.

```
var values = [1, 2, 3, 4, 5];
console.log(values.every(element => element % 2 === 0)); // false
console.log(values.every(element => element > 10)); // false

console.log(values.some(element => element % 2 === 0)); // true
console.log(values.some(element => element > 2)); // true
```

Arrays. Methods

- concat() - возвращает новый массив, который состоит из массива, в контексте которого метод был вызван, соединенный с массив(ом/ами) и/или другими значениями, которые были переданы как аргументы.
- slice() - возвращает мелкую копию части массива в новый массив, начиная с begin и до end (не включая end), где begin и end являются индексами элементов массива. Начальный массив не меняется.

```
console.log(['a', 'b', 'c'].concat(['d', 'e', 'f'])); // ['a', 'b', 'c', 'd', 'e', 'f']
console.log(['a', 'b', 'c'].concat(5)); // ['a', 'b', 'c', 5]

var animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];
console.log(animals.slice(2)); // ["camel", "duck", "elephant"]
console.log(animals.slice(1, 5)); // ["bison", "camel", "duck", "elephant"]
console.log(animals.slice(2, 4)); // ["camel", "duck"]
```

Arrays. Methods

splice() - универсальный метод для работы с массивами. Умеет все: удалять элементы, вставлять элементы, заменять элементы – по очереди и одновременно. Метод splice() изменяет состояние массива, удаляя и/или добавляя новые элементы.
splice(index, deleteCount, element1, ..., elementN) - сигнатура метода;

```
var arr = [“a”, “b”, “c”];
arr.splice(1, 1); // b console.log(arr) // [“a”, “c”];

var months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
console.log(months); // ["Jan", "Feb", "March", "April", "June"]
months.splice(4, 1, 'May');
console.log(months); // ["Jan", "Feb", "March", "April", "May"]
```

Arrays. Methods

sort() - сортирует массив на месте (это алгоритм, который преобразует входные данные без использования вспомогательной структуры данных) и возвращает отсортированный массив.

```
var values = [ 1, 2, 15 ];
console.log(values.sort()); // [1, 15, 2]
```

по умолчанию sort сортирует, преобразуя элементы к строке.

Для указания своего порядка сортировки в метод sort() нужно передать функцию fn от двух элементов (a и b), которая умеет сравнивать их.

Если эту функцию не указать, то элементы сортируются **как строки**.

Arrays. Methods

Алгоритм сортировки, встроенный в JavaScript, будет передавать ей для сравнения элементы массива. Функция должна возвращать:

- Положительное значение, если a > b,
- Отрицательное значение, если a < b,
- Если равны – можно 0, но вообще – не важно, что возвращать, если их взаимный порядок не имеет значения.

Массивы в JavaScript могут содержать в качестве элементов другие массивы.

```
var matrix = [  [1, 2, 3],  [4, 5, 6],  [7, 8, 9] ]; - многомерный массив.
console.log(matrix[1][1]); // 5
```

Arrays. Methods

Существуют и другие полезные методы в Array.prototype, которые могут пригодиться в повседневной жизни разработчика:

Array.prototype.copyWithin()
Array.prototype.entries()
Array.prototype.fill()
Array.prototype.join()
Array.prototype.keys()
Array.prototype.reverse()
Array.prototype.reduceRight()
Array.prototype.toLocaleString()
Array.prototype.toString()
Array.prototype.toSource()
Array.prototype.values()
Array.prototype.flatMap()

Arrays. Length

Длина length – не количество элементов массива, а последний индекс + 1.

```
var array = [];
array[10] = " ";
console.log(array.length); // 11
```

При уменьшении length массив сокращается.
Причем этот процесс необратимый, т.е. даже если потом вернуть length обратно – значения не восстановятся.

Самый простой способ очистить массив – это array.length = 0.

Objects (revise)

Объекты в JavaScript сочетают в себе два важных функционала.

Первый – это ассоциативный массив: структура, пригодная для хранения любых данных.

Второй – языковые возможности для объектно-ориентированного программирования.

Ассоциативный массив – структура данных, в которой можно хранить любые данные в формате ключ-значение.

В других языках программирования такую структуру данных называют «словарь» и «хэш».

Пустой объект может быть создан одним из трёх способов:

```
var object = new Object();
var object = Object.create(null);
var object = {};
```

Functions

Конструктор это функция которая будучи вызвана с ключевым словом **new** возвращает новый объект. Внутри конструкторов ключевое слово this указывает на новый создаваемый объект.

Опишем конструктор, который принимает на вход имя и просто присваивает его соответствующему свойству объекта:

```
var Human = function(name) {
    this.name = name;
}
```

Чем конструкторы отличаются от простых функций - ничем. Просто принято называть конструкторы начиная с большой буквы. Любая функция в языке JavaScript потенциально называется конструктором и любую функцию можно вызвать с ключевым словом **new**.

Functions

В каждой функции есть свойство prototype в котором хранится прототип свойства которого будут наследовать все объекты создаваемые при помощи конструктора. Изначально это практически пустой объект, но мы можем создавать в нём любые свойства и методы, которые в дальнейшем будут доступны любым экземплярам этого класса.

```
Human.prototype.anyMethod = function() {
    console.log(this.name + " do something...");
}

var man = new Human("Vasya");
var woman = new Human("Kasya");
console.log(man.name, " ", woman.name); // Vasya Kasya
man.anyMethod(); woman.anyMethod();
```

Functions

Важно понимать, что когда мы используем родные JavaScript конструкторы всем объектам, которые создаются при помощи конструктора - присваивается свойство constructor, также это свойство есть и на прототипе:

```
console.log(man.constructor); // function (name) {...}
console.log(Human.prototype.constructor); // function (name) {...}
```

Также в JavaScript есть бинарный оператор instanceof при помощи которого можно проверить любой объект на принадлежность к классу, класс определяется конструктором.

```
console.log(man instanceof Human); // true
console.log(Human.prototype.isPrototypeOf(man)); // true
```

Functions

То что, ссылка на прототип для новых объектов хранится внутри свойства prototype всех функций может немного усложнить понимание всего происходящего, поскольку услышав что функции в JavaScript являются объектами и что у всех объектов есть прототип, можно предположить, что:

Human.prototype // через свойство prototype доступен прототип функции Human? **НЕТ**

man.prototype // можно ли получить прототип объекта у экземпляров класса Human? **ТОЖЕ НЕТ**

Чтобы получить прототип объекта напрямую: есть свойство __proto__. Которое по разным причинам, к примеру поддержка разных браузеров, лучше без надобности не использовать.

Functions

Пример создания дочерних классов:

```
var Developer = function(name, skills) {
    Human.apply(this, arguments);
    this.skills = skills || [];
};
Developer.prototype = Object.create(Human.prototype);
Developer.prototype.constructor = Developer; // чтобы свойство constructor перестало ссылаться на функцию Human.

var developer = new Developer("Petya", ["JavaScript", "Node.js", "Java"]);
console.log(developer.name); // Petya
console.log(developer.skills); // ["JavaScript", "Node.js", "Java"]
developer.anyMethod(); // Petya do something...
```

Functions

Оператор instanceof будет возвращать **true** для всех классов у которых прототипы находятся в цепочке прототипов нашего объекта, другими словами:

```
console.log(developer instanceof Developer); // true
console.log(developer instanceof Human); // true
```

Необходимо понимать, что существуют также, так называемые, родные методы объектов. На вершине всей иерархии находится Object.prototype, другими словами все объекты в JavaScript наследуют свойства от объекта Object.prototype.

```
console.log(developer.toString()); // [object Object] - переопределим его
Human.prototype.toString = function() { return this.name };
```

Functions

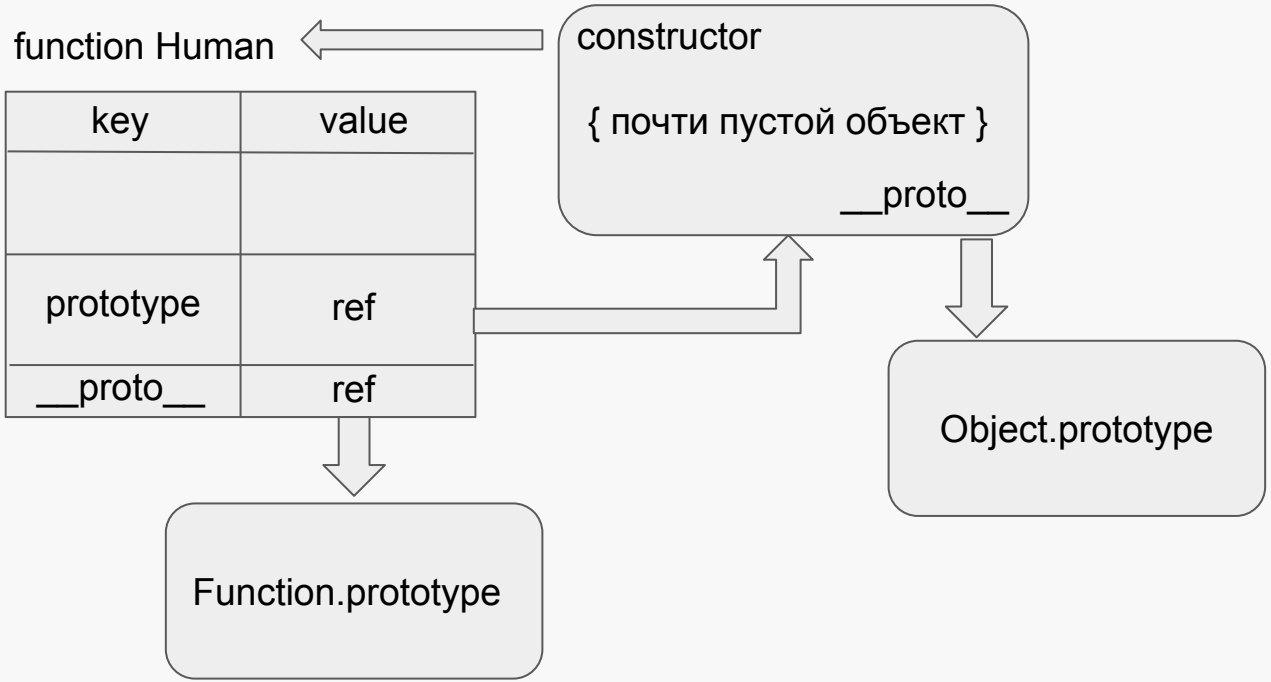
Разберёмся раз и навсегда как работает функция конструктор:

```
var Human = function (name, born, city) {
    // this = {} - неявно создаётся новый объект
    // this.__proto__ =  Human.prototype - тоже происходит неявно
    this.name = name;
    this.born = born;
    this.city = city;

    getAge = function() { return new Date().getFullYear() - this.born }
    // методу getAge место в свойстве prototype, поскольку он одинаков
    для всех, а в данном случае будет создаваться для всех.

    // return this - новый объект возвращается из функции
}
```

Functions


function Human
constructor
{ почти пустой объект }
__proto__
key
value
prototype
ref
__proto__
ref
Object.prototype
Function.prototype

Functions

```
Human.prototype.getAge = function() {
    console.log(new Date().getFullYear = this.born);
}
var user = new Human("Name", 1985, "Boston");
user.getAge() // 35
```

Главное помнить, что с появлением **ES6** данные конструкции стало писать намного легче, но - все методы которые теперь описываются в scope ключевого слова class - продолжают складываться в свойство prototype, а если нам нужно именно сделать так, чтобы какой-то метод стал свойством класса - нам необходимо сделать это по старинке - в конструкторе, используя ключевое слово this.

List. ArrayList

List - список, коллекция для хранения однотипных данных.

ArrayList - реализует интерфейс List. В Java массивы имеют фиксированную длину, и после того как массив создан, он не может расти или уменьшаться. ArrayList может менять свой размер во время исполнения программы, при этом не обязательно указывать размерность при создании объекта.

List. LinkedList

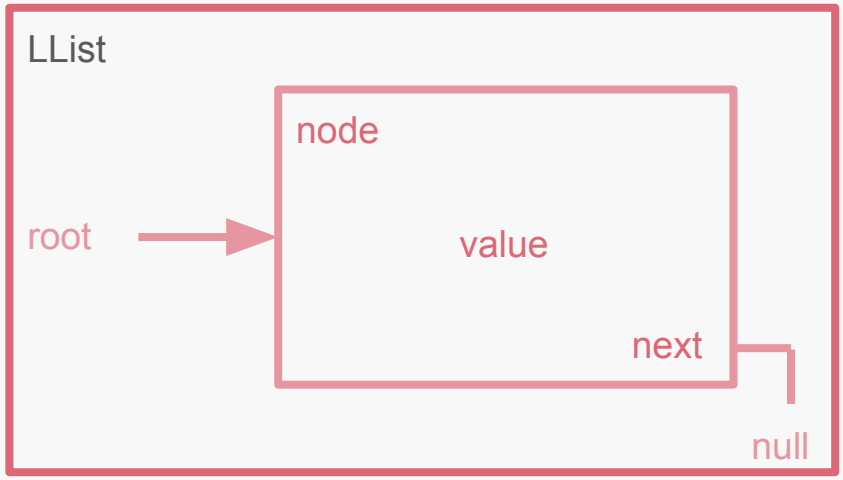
LinkedList - является представителем направленного списка, где каждый элемент структуры содержит указатель на следующий элемент. Реализует методы получения, удаления и вставки в начало, середину и конец списка.

List. LinkedList

Только что созданный объект list, содержит свойство root - указатель на начало списка.

Значение свойства root равно null при инициализации. Свойство next каждого контейнера элемента всегда указывает на следующий элемент списка. При инициализации контейнера оно также равно null.

List. LinkedList



List. Linked List (add elements)