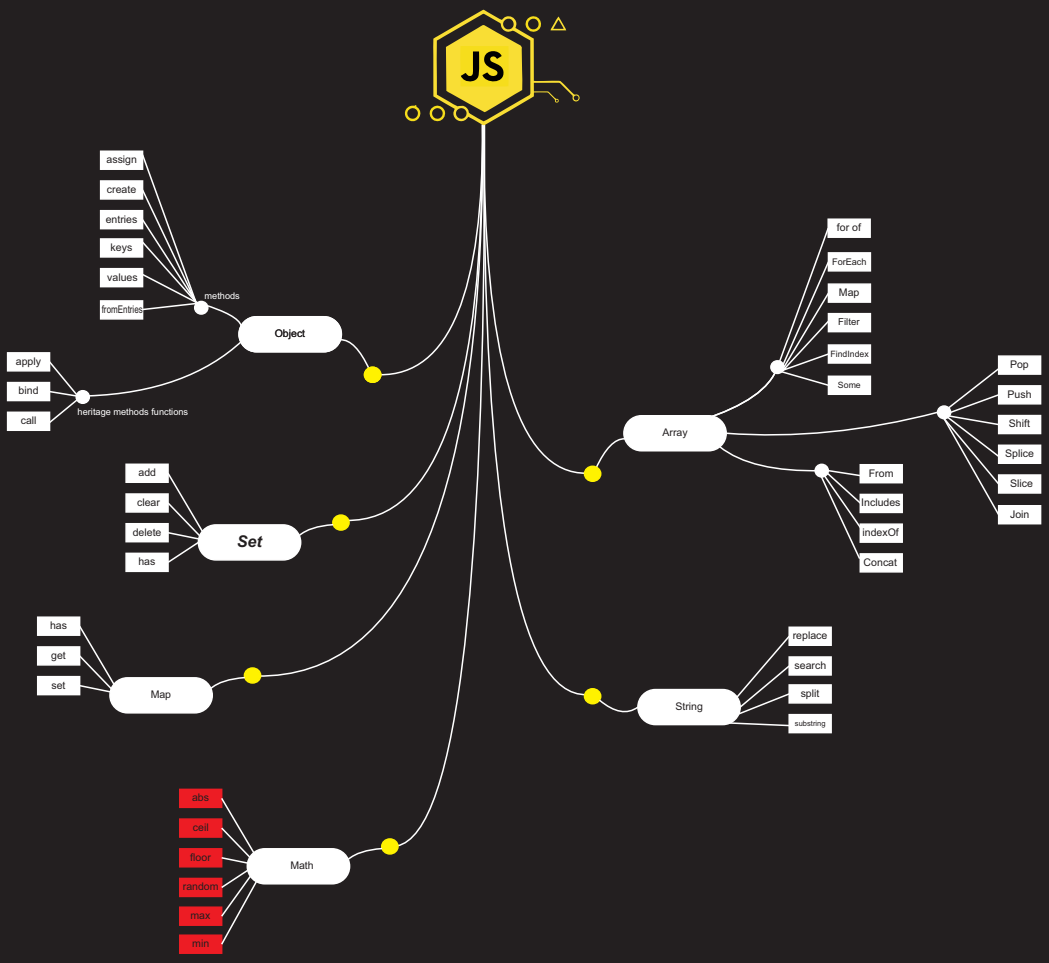
JS
Object
methods
assign
create
entries
keys
values
fromEntries
heritage methods functions
apply
bind
call
Array
for of
ForEach
Map
Filter
FindIndex
Some
Pop
Push
Shift
Splice
Slice
Join
From
Includes
indexOf
Concat
Set
add
clear
delete
has
Map
has
get
set
String
replace
search
split
substring
Math
abs
ceil
floor
random
max
min

For of

função mais próxima do nosso for padrão, este realizará iterações em todo array enviando o valor da posição atual para a posição em memória definida.

```
const myArr = [1, 2, 3];

for(let value of myArr){
  console.log(value);
}
//1
//2
//3
```

ForEach

este irá percorrer cada posição do array e enviar o valor da posição atual para um callback.

```
const myArr = [1, 2, 3];

myArr.forEach((value) => {
  console.log(value)
})
//1
//2
//3
```

Map

irá percorrer o Array executando e enviando o valor iterado para um callback. Para cada interação a função '.map' espera um retorno obrigatório de um valor, no final nos será devolvido um novo array de acordo com a manipulação. Geralmente utilizamos este método para formatar dados como mostra o exemplo abaixo.

```
const myUsers = [
  {
    name: 'Gabriel',
    money: '10'
  },
  {
    name: 'Oliveira',
    money: '50'
  }
]

const formatted = myUsers.map((u) => {
  u.money = u.money + 'R$';

  return u;
});
// [
```

FindIndex

O método ‘FindIndex’ percorrerá cada posição e o primeiro ‘callback’ que retornar um valor verdadeiro, será o índice a ser retornado do valor equivalente.

```
const myArr = [1, 2, 3];

const result = myArr.findIndex((i) => i === 3);
//() => 2
```

Find

O método ‘Find’ percorrerá cada posição e o primeiro ‘callback’ executado que retornar um valor verdadeiro, será retornado o seu valor da própria posição. Caso contrário nenhum valor será retornado.

```
const myArr = [1, 2, 3];

const result = myArr.find((i) => i === 3);
//() => 3
```

Filter

Um método muito parecido com os dois acima, a diferença é que nos será devolvido um array com todos os valores que sejam de acordo com a operação do ‘callback’. Caso não encontre nenhum será devolvido um array vazio.

```
const myArr = [1, 2, 3, 5, 6, 7, 8];

const result = myArr.filter((i) => i > 3);
//() => [5, 6, 7, 8]
```

Some

Um pouco parecido com o método ‘find/FindIndex’, este testa se um dos elementos no array está de acordo com a operação, se sim retorna ‘true’ caso encontre e ‘false’ caso não encontre.

```
const myArr = [1, 2, 3];

myArr.some((i) => i === 3);
// () => true
```

Pop

este remove o último elemento do array e retorna.

```
const myArr = [1, 2, 3];
const removed = myArr.pop();
// myArr = [1,2];
// removed = 3;
```

Push

O método push() adiciona um ou mais elementos ao final de um array e retorna o novo comprimento desse array.

```
const myArr = [1, 2, 3];
const newLength = myArr.push(4);
// myArr = [1, 2, 3, 4];
// newLength = 4;
```

Shift

O método shift()remove o primeiro elemento de um array e retorna esse elemento.

```
const myArr = [1, 2, 3];
const removed = myArr.shift();
// myArr = [2, 3, 4];
// removed = 1;
```

Splice

O método splice() altera o conteúdo de uma lista, adicionando novos elementos enquanto remove elementos antigos (ou simplesmente somente removendo elementos).
O seu retorno são elementos que foram (ou não) removidos.

Primeiro vamos tentar inserir elementos em determinada posição mantendo a integridade das posições seguintes.

```
const myArr = ["funny", "foo", "bill"];
const removed = myArr.splice(2, 0, "gaga");
//myArr = ["funny", "foo", "gaba" "bill"];
//removed = [];
```

Dizemos para função splice: a partir da posição 2 remova 0 elementos e insira a nossa string ‘gaga’.

apenas removendo elementos:

```
const myArr = ["funny", "foo", "bill", "gaga"];
const removed = myArr.splice(1, 1);
//myArr = [ 'funny', 'bill', 'gaga' ];
//removed = [ 'foo' ];
```

agora somente removemos 1 elemento a partir da posição 1 do nosso array.

Splice

O método slice() retorna uma cópia de parte de um array a partir de um subarray criado entre as posições início e fim (fim não é obrigatório) de um array original. O Array original não é modificado.

```
const myArr = ["funny", "foo", "bill", "gaga"];
const newArr = myArr.slice(1, 3);
//myArr = [ 'funny', 'foo', 'bill', 'gaga' ]
//removed = [ 'foo', 'bill' ]
```

From

Cria um novo array a partir de um array com a possibilidade de executar a função map para cada elemento do array

```
const myArr = [1, 2, 3];
const newArr = Array.from(myArr, (x) => x * 3);
//newArr = [ 3, 6, 9 ]
```

Includes

Determina se um array contém determinado elemento, retornando true ou false de acordo com a existência.

```
const myArr = [1, 2, 3];
const exist = myArr.includes(2);
// exist => true;
```

É possível determinar o índice inicial onde o método começará a buscar, basta informar a posição no segundo parâmetro..

IndexOf

Buscará determinado elemento dentro de um array, caso exista retornará a posição dele, se não retornará o valor numérico ‘-1'.

```
const myArr = [1, 2, 3, 5];
const p = myArr.indexOf(5);
//p = 3
```

Neste método também é possível informar a partir de qual índice queremos realizar a busca.

```
const myArr = [1, 2, 3, 5];
const p = myArr.indexOf(1, 1);
//p = -1;
```

Join

A função join pegará um array que irá ser convertida em uma string, sendo que um separador previamente definido é imposto entre cada elemento.

```
const arr = ['Ga', 'Ba', 'Ri', 'El'];
const myJoin = arr.join('|');
//myJoin = Ga|Ba|Ri|El;
```

Object

Create

O método Object.create() cria um novo objeto, utilizando um outro objeto existente como protótipo para o novo objeto criado.

```
const lastObject = {
  a: 1,
  b: 2
}

const newObject = Object.create(lastObject);

console.log(newObject.__proto__);
// newObject.__proto__ = { a: 1, b: 2 }
```

Ao subirmos na cadeira de protótipo repare que obtivemos o objeto ‘lastObject’ que podem ser acessados a partir de newObject.

Assign

Este método é utilizado para copiar valores de todas as propriedades de um objeto para um objeto destino.

```
const obj1 = { a: 1, b: 100 };
const obj2 = { b: 2, c: 3 };
const newObj = Object.assign(obj1, obj2);
console.log(newObj);
// newObj = { a: 1, b: 2, c: 3 }
```

repare que o a propriedade do objeto de destino (obj2) ‘b’ não foi sobrescrita pois a função considera essa como prioridade.

Entries

Retornará um array aninhado com todos os pares do objeto. [chave, valor] na mesma ordem dos objetos.

```
const obj = {
  a: 1,
  b: 2,
  c: 3
}
const entries = Object.entries(obj);
// entries = [ [ 'a', 1 ], [ 'b', 2 ], [ 'c', 3 ] ]
```

Object

Keys

Este irá criar um array único onde cada posição irá representar a chave correspondente dentro do objeto:

```
const obj = {
  a: 1,
  b: 2,
  c: 3
}
const entries = Object.keys(obj);
// entries = [ 'a', 'b', 'c' ];
```

Values

Este irá criar um array único onde cada posição irá representar o valor da chave correspondente dentro do objeto:

```
const obj = {
  a: 1,
  b: 2,
  c: 3
}
const entries = Object.keys(obj);
// entries = [ 1, 2, 3 ];
```

FromEntries

O método Object.fromEntries() transforma uma lista de pares chave-valor em um objeto.

```
const entries = new Map([
  ['foo', 'bar'],
  ['baz', 42]
]);
const obj = Object.fromEntries(entries);
// { foo: "bar", baz: 42 }
```

Apply

O método apply invoca a função que está sendo acessada o método, informando o valor 'this' que a função irá considerar. (todos os argumentos devem ser enviados em um array)

```
const obj = {
  value: 1,
  calc(n, y, x){
    return this.value + n + y + x;
  }
}
const newObj = {
  value: 1000
}
obj.calc.apply(newObj, [25, 3, 8]);
//1036
```

Call

A função 'call' realiza o mesmo trabalho da função 'apply' sob a diferença que podemos mandar os argumentos separadamente em vez de enviar um array.

```
const obj = {
  value: 1,
  calc(n, y, x){
    return this.value + n + y + x;
  }
}
const newObj = {
  value: 1000
}
obj.calc.call(newObj, 25, 3, 8);
//1036
```

Bind

O método bind tem uma função similar as funções 'call' e 'apply', sendo sua diferença que ao invés de executar imediatamente a função nos é retornada a referência dela.

```
const obj = {
  number: 10,
  calc(value){
    return this.number * value;
  }
}
const obj2 = {
  number: 20,
}
const func = obj.calc.bind(obj2);
func(10);
//200
```

Replace

O método ‘replace’ pegará a primeira ocorrência da ‘substring‘ informada e substituirá pelo novo conjunto informado. Sendo possível trabalhar com expressões regulares e manipular o método replace informando um callback.

```
const myString = 'Javascript, Python, Javascript';
const stringReplace = myString.replace("Javascript", "C++");
//myString = 'Javascript, Python, Javascript';
//stringReplace = 'C++, Python, Javascript';
```

Search

método muito similar ao ‘indexof’ sendo que a função deste é encontrar o índice de uma expressão regular informada dentro da string a ser manipulada.

```
let str = "hey.JudE";
let reDot = /[.]/g;
let position = str.search(reDot);
//position = 3;
```

Split

A função ‘split’ converterá uma string em um array como base em um separador entre cada conjunto de sub string’s.

```
const string = 'Hello i am a code of Javascript';
const separator = ' ';
const split = string.split(separator);
//separator = [ 'Hello', 'i', 'am', 'a', 'code', 'of', 'Javascript' ];
```

Substring

O método substring() retorna a parte da string entre os índices inicial e final, ou até o final da string.

```
var anyString = "Mozilla";
var mySubString = anyString.substring(0,3);
//mySubString = Moz
```

Set

O objeto Set permite que você armazene valores únicos de qualquer tipo, desde valores primitivos a referências a objetos.
Objetos Set são coleções de valores nas quais é possível iterar os elementos em ordem de inserção.

```
const mySet = new Set([1, 2, 3]);
//mySet = Set { 1, 2, 3 }
```

Add

O método add() acrescenta um novo elemento com o valor especificado no final de um objeto Set.

```
const mySet = new Set([]);
mySet.add(1);
mySet.add(true);
mySet.add('i am a string');
// mySet =  Set { 1, true, 'i am a string' }
```

Clear

O método clear() remove todos os elementos de um objeto Set.

```
const mySet = new Set([]);
mySet.add(1);
mySet.add(true);
mySet.clear();
// mySet =  Set {    }
```

Has

O método has() retorna um valor booleano indicando se um elemento com o valor especificado existe em um objeto Set ou não.

```
const mySet = new Set([]);
mySet.add(1);
mySet.add(true);
mySet.add('i am?');
mySet.has(1);
//true
mySet.has(true);
//true
mySet.has('i am?');
//true
```

Set

Delete

O método delete() remove o elemento especificado de um objeto Set.

```
const mySet = new Set([1]);
mySet.delete(1);
//mySet = Set {}
```

Map

O objeto Map é um mapa simples de chave/valor. Qualquer valor (objeto e valores primitivos) pode ser usado como uma chave ou um valor. Um objeto Map itera seus elementos em ordem de inserção — um loop for...of retorna um array de [chave, valor] para cada iteração.

```
const myMap = new Map();
```

Set

O método set() adiciona ou atualiza um elemento com uma chave e valor específicos a um objeto de Map.

```
var myMap = new Map();
myMap.set('bar', 'foo');
myMap.set(1, 'foobar');
// myMap = Map { 'bar' => 'foo', 1 => 'foobar
// Atualiza um elemento no map
myMap.set('bar', 'baz');
// myMap = Map { 'bar' => 'baz', 1 => 'foobar' }
```

Get

O método get() retorna um elemento específico de um objeto de Map.

```
var myMap = new Map();
myMap.set('bar', 'foo');
myMap.get('bar');
// foo
```