O que é um algoritmo

Algoritmo é uma **sequência de passos** (instruções) que serve para fazer algo.

Exemplos de algoritmo na vida real:

- Como sair de casa e chegar no shopping e
- Se vestir.

O que é uma Estrutura de Dados

Primeiro, temos que entender que **dados** são diferentes de uma **Estrutura de Dados.**

A função de uma **Estrutura de Dados** é *organizar* e *administrar* os seus **dados**.

Achei importante colocar:

FIFO - First in First Out
LIFO - Last in First Out

Organização Básica de um código Javascript

Um código **Javascript** é organizado em *sentenças de códigos* e *blocos de código*.

Uma **sentença e um bloco de código**:

```
console.log(`Oi,
pessoa!`);

{
   console.log(`Oi, pessoa!`);
}
```

Comentários

Observação: Sempre tente apenas fazer comentários relevantes e evite fazer comentários irrelevantes.

```
// Comentário de apenas uma linha
```

```
/*
   * Comentário de
   * múltiplas linhas
*/
```

O Básico de Var, Let e Const.

Var

Pode ser declarada mais de uma vez e o seu valor pode ser mudado.

Let

Pode ser declarada apenas uma vez e o seu valor pode ser mudado.

Const

Pode ser declarada apenas uma vez e o seu valor não pode ser mudado.

Observações

- Sempre que for usar uma variável que irá guardar um valor que não será alterado ou um *objeto*, *array*... Use o Const.
- Se for usar uma variável que o seu valor pode ser mudado, use o Let ao invés do Var.

Tipagem Fraca

O Javascript é de uma tipagem fraca (dinâmica).

Os dados dentro do Javascript tem tipo, mas você não consegue forçar o programador a colocar dados de apenas um tipo dentro de uma variável.

Vendo o tipo de uma variável ou dado

```
console.log(typeof "Oi");
```

```
→ string
```

Tipo Number

Criando um número a partir de uma função.

```
const peso2 = Number(2.0);
```

Verificando se um dado ou variável é um número inteiro

```
console.log(Number.isInteger(peso1)); → true
```

Formatando as casas decimais de um número

```
const media = 10;

console.log(media.toFixed(2)); → 10.00
```

Retornando um número do tipo number em string.

```
console.log(media.toString());
```

Retornando o valor em binário de um número em string.

```
console.log(media.toString(2)); → 1010
```

Alguns Cuidados do tipo Number em Javascript

A divisão por Zero retorna Infinity.

```
console.log(7 / 0); → Infinity
```

Dividindo um número do “tipo” string.

No Javascript, é possível fazer operações matemáticas em um número que seja do tipo string. O Javascript faz a conversão deste número para number. (Se o número dentro das aspas for válido).

```
console.log("7" / 2); → 3.5
```

Concatenando ao invés de somar

```
console.log("3" + 2); → 32
```

Não é possível fazer operações matemáticas em uma string.

Em algumas linguagens de programação, é possível realizar algumas operações “matemáticas” em uma string. Como repetir essa string por 2 vezes.

```
console.log("Show!" * 2); → NaN (Not a Number).
```

Erro ao fazer algumas operações em um número literal.

Número literal: é um valor “direto” (Como se não estivesse em uma variável)
Exemplo: “Oi”.concat(“nome”)

Você não conseguirá fazer algumas operações em cima de um número literal sem casas decimais em Javascript da forma normal, mas tem um jeito de se fazer, que é colocando o número literal dentro de um par de parênteses.

```
console.log(10.toFixed(2)); → Retorna um erro. (SyntaxError: Invalid or unexpected token)
```

Jeito certo de se fazer:

```
console.log((10).toFixed(2)); → 10.00
```

Ou

```
console.log(10.00.toFixed(2)); → 10.00
```

Math e API

O Javascript tem uma série de objetos úteis dentro dele, que são chamados de API (**Application Programming Interface** (Interface de programação de aplicações)).

Um desses objetos é o Math.

```
console.log(Math.floor(2.17)); → 2

console.log(typeof Math); → object
```

Tipo String

```
const escola = "Cod3r";
```

Retornando o caractere que está em um índice (Retorna um espaço em branco se o índice não existir)

```
console.log(escola.charAt(4)); → r
```

Retornando o valor do caractere na tabela UNICODE/ASCII

```
console.log(escola.charCodeAt(4)); → 114
```

Retornando o índice no qual o caractere está.

```
console.log(escola.indexOf("r")); → 4
```

Retornando os caracteres que estão no intervalo dos índices.

```
console.log(escola.substring(0, 2)); → Co (O caracter que está no índice 2 não será incluído)
```

Concatenando strings.

```
console.log("Escola".concat("!")); → Escola!
```

Substituindo caracteres de uma string.

```
console.log("Escola".replace("Es", "Coca - ")) → Coca - cola
```

Dividindo uma string e a transformando em um array de acordo com o caractere utilizado para a divisão.

```
console.log("Nicolas, Robert, Roberto, Maria".split(",")); → [ 'Nicolas', ' Robert', ' Roberto', ' Maria' ]
```

Tipos em Javascript: Boolean

Transformando um valor em **true** ou **false**.

Pegando o valor **original (Símbolo de negação)**

```
console.log(!!true); → true
```

Pegando o valor **inverso (Símbolo de negação)**

```
console.log(!true); → false
```

Valores que são verdadeiros.

- Todos os números inteiros são verdadeiros, menos o número 0. {10, 200, -200, -1 .
- Array ou um objeto **vazio** é verdadeiro.
- O valor Infinity também é verdadeiro.

Valores que são falsos.

- Uma string vazia é falsa.
- Undefined, null, NaN são falsos.

Retornando o primeiro valor verdadeiro entre X valores.

```
console.log(("" || 0 || "Ola")); → Ola → Retorna "Ola", pois este valor é true.
```

```
console.log(!!("" || 0 || "Ola")); → true → Retorna true, pois pelo menos um dos valores dentro dos colchetes é true, que nesse caso é a string "Ola".
```

Setando um valor para uma variável se ela estiver vazia.

```
let nome = "";

console.log(nome || "Desconhecido"); → Desconhecido → Retorna
“Desconhecido”, pois a variável “nome” está vazia.
```

Array

Um array dentro do Javascript é um objeto (object), assim, você poderá usar a notação ponto.

O que é um Array?

Um Array (ou vetor), é uma estrutura indexada que serve para agrupar múltiplos valores a partir de um único identificador.

Como o Javascript é de tipagem fraca, o Array no Javascript não tem tipo e nem um tamanho fixo.

Objeto

O que é um Objeto?

Um objeto é uma coleção de pares de chave e valor.

Imprimindo um objeto em forma de tabela

```
console.table(objeto);
```

Declarando um Objeto de forma literal.

```
const produto = {};

produto["Desconto legal"] = 0.40; Evite dar nome a um atributo com
espaço.

console.log(JSON.stringify(prod2)); >> Transforma um objeto em JSON.
```

Null e Undefined e valor passado por referência ou por valor

Valor passado por referência ou por valor

Referência

Quando uma variável que tem um objeto, array ou uma function é “copiada”, o valor que será passado para a variável que irá realizar a "cópia" é o endereço de memória e não o valor. Assim, quando você faz uma mudança em uma variável, a outra também irá mudar, pois elas estão no *mesmo endereço de memória*.

Valor

Quando uma variável que tem um valor primitivo é “copiada”, o valor que estiver dentro dela será copiado para a variável que está realizando a "cópia", e cada uma irá apontar para um endereço de memória diferente.

Undefined e Null

Undefined

É quando uma variável foi declarada mas não foi inicializada. Não atribua o valor undefined para uma variável, deixa que a linguagem declare quando você necessário.

Null

É quando uma variável foi inicializada mas ela não aponta para nenhum endereço de memória. O valor null pode ser usado para zerar uma variável do tipo referência (Object, Array, Function). Se for um valor primitivo, você pode colocar o valor 0.
Exemplo: Você pode zerar uma variável que aponta para um função, objeto, array.. com o null.

valor.toString(); → Irá dar um erro, pois os valores null e undefined não tem a notação ponto.

Função

A função é muito importante no Javascript.

Function, Object, classe são **funções**.

Função anônima (sem nome) armazenada em uma variável.

```
const imprimirSoma = function(num1, num2) {
   console.log(num1 + num2);
};
```

Função arrow

```
const soma2 = (num1, num2) => {
   return num1 + num2;
}
```

Função arrow com retorno implícito (apenas uma sentença).

```
const soma = (num1, num2) => num1 + num2;
```

Declaração de variáveis com Var

O escopo do Var é *global* e de *função*.

Quando uma variável é declarada com a palavra var e estiver fora de uma função, ela fará parte do objeto window, então, tente sempre fugir do escopo global para uma variável não sobrescrever a outra que tiver o mesmo nome.

Declaração de variáveis com Let

O escopo do Let é *global*, de *função* e de *bloco*. Uma variável declarada com a palavra let não fará parte do objeto window.

Var e let em loop

Quando a variável contadora é criada com a palavra Var, ela ficará visível globalmente, pois o var **não tem escopo de bloco**. Com a palavra let não iria dar para acessar a variável contadora fora do bloco do for.

```
for(var i = 0; i < 10; i++) {
   console.log(i);
};

console.log(i); → Vai retornar 10, pois 10 é o valor que a condição do
for ficou falsa.
```

Var não tem memória do valor

```
const array = [];

for(var i = 0; i < 10; i++) {
   array.push(function() {
       console.log(i);
   });
};

array[0](); → 10, pois a palavra Var não tem uma memória de qual era o
valor de i naquele momento, já com a palavra let, o valor seria 0.
```

Hosting (içamento)

Hoisting é um comportamento padrão do Javascript de mover a declaração de variáveis com a palavra Var para o topo do programa. **Não é indicado.**

```
console.log(teste); → undefined, mas ela está criada no programa.

var teste = "Teste";

console.log(teste); → Teste
```

Quando você declara a variável com o Var, ela é içada para o topo do programa e declarada com o valor undefined, ela só será inicializada quando o programa estiver na linha que você iniciou ela.

Função e objeto

No Javascript, a função faz o papel de uma classe, mas é possível criar uma classe com a palavra “class”.

```
console.log(typeof new Object); → É um objeto criado a partir de uma função → object
```

```
const Cliente = function() {};
console.log(typeof Cliente); → function
console.log(typeof new Cliente); → object
```

```
console.log(typeof class Produto {}); → function, pois o "class" é um atalho de sintaxe e é convertido para função.

console.log(typeof new class Produto {}); → object
```

Par nome/valor

Observação: Objetos são grupos aninhados de pares nome/valor

```
const nome = "Nicolas"; → Está é uma estrutura de par de chave e valor. chave = nome e o valor é “Nicolas”.
```

Contexto léxico

Contexto léxico é onde sua variável foi definida fisicamente no *código*.

```
const saudacao = "Opa";

function exec() {
   const saudacao = "Fallaa."; → Este é um outro contexto léxico, por
isso não irá dar um erro..
};

const saudacao = "Ela"; → Vai dar erro, pois já existe uma constante
com o nome saudacao neste contexto léxico.
```

Criando uma classe a partir de uma função

```
function Obj(nome, idade) {
   this.nome = nome; → Criando um atributo.
   this.idade = idade;
   this.falar = () => {
       console.log("Falando...");
   }; → Criando um método.
};
```

Operadores de atribuição

Atribuição de acrescentação → **+=**

Atribuição subtrativa→ **-=**

Atribuição multiplicativa→ ***=**

Atribuição divisiva → **/=**

Atribuição modular→ **%=**

Operadores: Destructuring

É um operador de desestruturação, ele tira algo de uma estrutura.

- Array: Tira os seus elementos. O sinal de [ ] é usado.
- Objeto: Tira os seus atributos. O sinal de { } é usado.

Os sinais [ ] e { } sinalizam a operação de desestruturação.

Com objeto

```
const pessoa = {nome: "Nicolas", idade: 16};

const {nome, idade} = pessoa;

console.log(nome, idade); → Nicolas 16
```

Dando um nome diferente a variável

Nesta forma, temos que colocar o nome de nossas variáveis exatamente igual ao nome do atributo que queremos tirar, para colocarmos o nome da variável diferente do nome do atributo, temos que usamos esta forma:

```
const pessoa = {nome: "Nicolas", idade: 16};

const {nome: n, idade: i} = pessoa;

console.log(n, i);
```

Setando um valor padrão, para que se o atributo não exista, ele fique com o valor setado.

```
const pessoa = {nome: "Nicolas", idade: 16};

const {nome, idade, bemHumorada = true} = pessoa;

console.log(nome, idade, bemHumorada); → Nicolas 16 true
```

Pegando um valor dentro de um objeto.

```
const pessoa = {nome: "Nicolas", idade: 16, endereco: {logradouro:
"Logradouro 123", numero: "234"}};

const {endereco: {logradouro, numero, cep}} = pessoa;

console.log(logradouro); → Logradouro 123
```

Não dá erro com a variável cep (ela não está declarada dentro do objeto endereço), pois ela é a última a ser declarada.

Com array

```
const [n1, , n3, , n5, n6 = 0] = [10, 7, 9, 8];
```

, = A vírgula serve para pular o elemento que está naquela posição.

n6 = 0 = Seta um valor padrão para a variável n6, para que se não tenha nenhum elemento naquele índice, o valor fique como 0;

```
const [, [, nota]] = [[, 8, 8], [9, 6, 8]];

console.log(nota); → 6
```

Com função (passando parâmetros)

Objeto

```
function rand({min = 0, max = 1000}) {
   const valor = Math.random() * (max - min) + min;
   return Math.floor(valor);
};

console.log(rand({min: 50, max: 100}));

console.log(rand()); → Quando o valor passado para a desestruturação é null ou undefined, o programa lança um erro, pois ele está tentando desestruturar "nada".

{min = 0, max = 1000} = {} → Fazendo assim, não dará erro, pois você está definindo um objeto vazio como um valor padrão.
```

Array

```
function rand([min = 0, max = 1000] = []) {
   if (min > max) [min, max] = [max, min]; → Verifica se o mínimo é maior que o máximo e se for, troca os números de posição (O mínimo vai para a variável max e o máximo para a variável min).

   const valor = Math.random() * (max - min) + min;
   return Math.floor(valor);
};

console.log(rand([40, 50]));
```

Operadores aritméticos

Os operadores aritméticos são operadores binários (operam em cima de dois operandos)

```
const soma = a + b + + c; → O resultado de a + b é somado com o valor de c. Por isso que são chamados de operadores binários.
const subtracao = a - b;
const multiplicacao = a * b;
const divisao = a / b;
const modulo = a % b;
```

Post fix (Depois do operando)

```
c++;
```

Pre fix (Antes do operando)

```
++c;
```

In fix

```
a + b;
```

Operador unário (Apenas um operando)

```
-a;
```

O parenteses tem preferência

```
c + (a + b); → A primeira operação a ser feita vai ser a de (a + b) e depois irá somar o resultado dessa operação com o valor de c.
```

Operadores: Relacionais

Uma operação **relacional** sempre vai retornar true ou false.

```
console.log(1 == "1"); Compara a igualdade de valor entre dois valores.

console.log(1 === 1); Compara a igualdade de valor e de tipo entre dois valores.

console.log(1 != "1"); → Compara a desigualdade de valor entre dois valores.

console.log(1 !== 1); → Compara a desigualdade de valor e de tipo entre dois valores.

      > Maior que
      < Menor que
      >= Maior igual que
      <= Menor igual que
```

```
const d1 = new Date(0);
const d2 = new Date(0);

console.log(d1 == d2); → Tá comparando o endereço de memória, pois d1 e d2 são objetos.

console.log(d1.getFullYear() == d2.getFullYear()); → Compara o valor, pois getFullYear() retorna um número.
```

Operadores lógicos

&& (e) → Todos os operandos tem que ser verdadeiros.

|| (ou) → Pelo menos um dos operandos tem que ser verdadeiro.

xor (ou exclusivo) → Sempre um operando tem que ser diferente do outro.

Operador xor tabela

Verdadeiro e falso, Falso e verdadeiro → Verdadeiro

Falso e falso, verdadeiro e verdadeiro → Falso

Esses *três operadores lógicos* que o Javascript possui, são operadores binários. E o *operador de negação (!)* é um operador unário, pois atua em cima de apenas um operando.

```
function compras(trabalho1, trabalho2) {
 const comprarSorvete = trabalho1 || trabalho2;
 const comprarTv50 = trabalho1 && trabalho2;

 // const comprarTv32 = !!(operando1 ^ operando 2) bitwise xor
 const comprarTv32 = trabalho1 != trabalho2; operador xor com
gambiarra.

 const manterSaudavel = !comprarSorvete;
 return {comprarSorvete, comprarTv50, comprarTv32, manterSaudavel};
};

console.log(compras(true, false)); → {

            comprarSorvete: true,
            comprarTv50: false,
            comprarTv32: true,
            manterSaudavel: false
      }
```

Operadores unários

```
let num1 = 1;
let num2 = 2;

num1++;

console.log(num1); → 2

--num1; → A pre fix tem uma prioridade maior que a post fix.

console.log(++num1 === num2--); true → Na comparação, o num1 está com o valor 2 e o num2 também, pois a operação de diminuir uma unidade de num2 foi feita depois da comparação. Não use incrementos em uma comparação, pois o código não será facilmente lido.
```

Operador ternário

Um operador ternário começa com uma expressão lógica. Se essa **operação** for **verdadeira**, volte a primeira "Frase", **senão**, volte a segunda "Frase".

```
const resultado = nota => nota >= 7 ? "Aprovado" : "Reprovado";

console.log(resultado(7)); → Aprovado
console.log(resultado(5)); → Reprovado
```

Try Catch Throw

```
function tratarErroELancar(e) {
   //- O throw serve para lançar um erro, uma mensagem, um objeto...

   throw new Error("Um erro aconteceu...");
};
```

```
function imprimirNomeGritado(obj) {
  try { → Tente executar isso
   console.log(obj.name.toUpperCase() + "!!!");
 }
  catch(e) { → Não mande detalhes da infraestrutura ou um erro padrão
da linguagem para o usuário, e sim, uma mensagem que informe o que está
acontecendo ou uma mensagem genérica.
   tratarErroELancar(e);
 }

 finally { → Executa as instruções passadas, mesmo se ocorrer um erro.
   console.log("Programa finalizado!");
 };
};

const obj = {nome: "Roberto"};
imprimirNomeGritado(obj);
```

Contexto de Execução: Browser vs Node

This

Dependendo do runtime (local) que você está executando, o this pode variar.

No **navegador** → objeto **window** →Na web, todos os arquivos **.js** vão todos para o objeto **window**.

No **Node** → module.exports →Cada arquivo no Node, é um **módulo**.

O objeto “window” do Node.

O objeto “**window**” do Node é o **objeto global**. **Variáveis criadas** com (*var*, *let* e *const*) **não** vão para o **objeto global**, **apenas** variáveis criadas sem **nenhuma** dessas palavras **reservadas**.

```
a = 10;

console.log(global.a); → 10
```

Funções

Função com **nome** ou uma função **anônima** vão para o objeto window.

```
function f1 () {
   console.log(this); Irá mostrar o objeto window (web) ou o objeto
global (Node).
};
```

Arrow function

Uma **arrow function** *só* irá para objeto **window** (web) se ela for declarada com **var** ou **sem uma palavra** reservada.

Objeto

```
let pessoa = {
   nome: "Ana",
   falar: function() {
       return "Eu sou " + this.nome; → O this se refere ao objeto
pessoa e não ao objeto window.
   }
};
```

aa