Algoritmos e Lógica de Programação

Quando a necessidade é desenvolver um programa ou rotina a ser executada pelo computador, precisamos deixar bem claro a sequência que deve ser seguida para atingir o resultado esperado. A esse encadeamento lógico na programação, chamamos de lógica de programação, e a descrição de como fazer, definimos como algoritmos.

Algoritmos são sequências de passos que seguimos com a intenção de atingir um objetivo

Na lógica de programação é onde aplicamos todos os conceitos de algoritmos, a definição do passo a passo e transferimos toda a lógica do algoritmo desenvolvido para uma linguagem de programação.

Em linhas gerais, lógica de programação é todo conjunto de regras e conceitos que precisamos aplicar para criar códigos que serão interpretados e executados por um computador.

**Fundamentos JS: Tipos, variáveis e funções**

**TIPOS DE DADOS**

**Tipos primitivos**

*Números* - servem para armazenar números;

Números decimais – chamados de ponto flutuante

São declarados normalmente como nºs e no momento de associar um valor, coloca-se o ponto seguido do valor. Pontos flutuantes também podem ser representados usando notação exponencial: um número real seguido pela letra e (ou E), seguido por um sinal opcional de mais (+) ou menos (-), e por um expoente inteiro.

Divisão por zero não é um erro em JavaScript: ele simplesmente retorna “Infinity”. No entanto, há uma exceção: zero dividido por zero não tem um valor bem definido e o resultado dessa operação é o valor especial não numérico NaN.

Dica: podemos utilizar o número [PI](https://pt.wikipedia.org/wiki/Pi) através com o código Math.PI.

No caso de números, é possível salvar em uma variável não somente o número em si, como em const num = 5;, mas também o resultado de uma operação (ou mais):

const soma = 10 + 10;

const multiplicacao = 10 \* 10;

const operacao = (soma + multiplicacao) + 10;COPIAR

FORMATANDO NºS

### método Math.round() retorna o valor de um número arredondado para o inteiro mais próximo.

### Formatando o resultado com ajuda do método toFixed()

### Controla o nº de casas decimais após vírgula. Outro ponto importante de se observar é que seu retorno será uma **string** representando o número

### Formatando o resultado para Real

método chamado toLocaleString() que converte um número para uma string, já tratando a questão do arredondamento e convertendo para a moeda do país que queremos, no nosso caso, o real, tornando a tarefa do programador muito mais simples.

O método **toLocaleString()** recebe alguns argumento - um objeto literal com as propriedades -, no meu caso eu utilizei:

* style : Que é o estilo do formato a ser utilizado, aqui é permitido usar:
  + decimal para representar números simples.
  + currency que diz respeito ao formato monetário e que vai indicar a moeda que vai ser utilizada.
  + percent para formato percentual.
* currency: A moeda para usar na formatação monetária

const formatado = salarioHora.toLocaleString('pt-BR', { style: 'currency', currency: 'BRL' });

No JavaScript temos outros métodos que podem ser utilizadas para o arredondamento como:

* **Math.ceil()** que retorna o maior número inteiro que é maior que o número passado, por exemplo Math.ceil(11.123), o valor fica 12
* **Math.floor()** que retorna o menor número inteiro que é menor que o número passado, por exemplo Math.floor(11.789), o valor fica 11

Strings - servem para armazenar textos e caracteres;

As strings conseguimos usar no JavaScript para guardar tipo texto, ou seja, tudo que não for número ou não for booleano

**Trabalhando com strings**

const cidade = "belo horizonte";

const input = "Belo Horizonte";

const inputMinusculo = input.toLowerCase();

console.log(cidade === inputMinusculo); // trueCOPIAR CÓDIGO

Acima, vemos um dos **métodos de string nativos do JavaScript** em ação, o toLowerCase() que converte todos os caracteres da string informada (no caso, input) para letras minúsculas (se forem algarismos, nada é convertido

Outro exemplo: qualquer inserção de texto que exija uma quantidade mínima de caracteres, como uma senha ou um nome. A propriedade length pode ser utilizada para sabermos quantos caracteres uma string contém:

const senha = "minhaSenha123"

console.log(senha.length)

*B*ooleanos - servem para armazenar verdadeiros ou falsos.

Turthy ou falsy –

// null => vazio

// undefined => Variável não definida

// 0 => false

// 1 => true

O null é um tipo especial, pois pode ser traduzido como “ausência de valor” e pode ser atribuído como valor de uma variável:

Nesse caso, qual seria a diferença entre os dois casos abaixo?

let input = null;

let input2;

console.log(input); // null

console.log(input2); // undefined

É aqui que entra o tipo undefined. Este tipo também representa “ausência de valor”, porém de uma outra forma: usualmente, enquanto null é um valor atribuído a uma variável que existe e foi iniciada, undefined se refere ao valor de uma variável que não foi inicializada (ou seja, não foi atribuído nenhum valor a ela).

undefined também é o valor retornado por uma função que não tem cláusula return.

**NAN** – Not a Number

Quando o JS tenta fazer uma operação mas não consegue pq o resultado da operação não é reconhecido como número.

**VARIÁVEIS**

Quando um programa precisa reter um valor para uso futuro, ele atribui o valor a (ou “armazena” o dado em) uma variável. As variáveis têm **nomes** e permitem o uso desses nomes em nossos programas para se referir a valores.

Os tipos de JavaScript podem ser divididos em duas categorias: tipos **primitivos** e tipos de **objetos**. Os tipos primitivos do JavaScript incluem números, palavras ou texto (conhecidas como **strings**) e valores booleanos (conhecidos como **booleanos**).

# Ordenação de números no JavaScript não funciona?

<https://www.alura.com.br/artigos/ordenacao-de-numeros-no-javascript-nao-funciona>

CONVERSÕES DE TIPO

Implícita A Conversão Implícita permite que nós convertamos um tipo de dado em outro

Podemos converter um number ou um número em uma string, uma string em um número, entre outras possibilidades.

*const* numero = 456;

*const* numeroString = "456";

console.log (numero === numeroString) // compara valor e tipo

console.log (numero == numeroString) // compara apenas o valor

// conversão explícita

// Number()

// String()

console.log (numero + numeroString); // concatena

console.log (numero + *Number*(numeroString)) // converte a string em number e realiza a soma

explícita

CONSOLE

O console é uma ferramenta que tem tanto no Node, que estamos usando, quanto nos navegadores e usamos ele para colocar frases para fora, os dados, de uma forma geral, para fora da nossa aplicação. De uma forma que não apareça, diretamente, para o nosso usuário, mas, ele tem acesso, caso ele precise.

O que é esse “log”, no final? É um registro. Estamos criando um registro no nosso console e podemos colocar qualquer informação dentro desse registro, podendo ser um número, uma string, uma variável, o que quisermos.

 Já vimos a stacktrace, que é resultado de um erro quando ele é chamado do jeito certo, que vamos ver mais para frente.

ntre os outros métodos, existem:

* console.error() para exibir mensagens de erro;
* console.table() para visualizar de forma mais organizada informações tabulares;
* console.time() e console.timeEnd() para temporizar período que uma operação de código leva para ser iniciada e concluída;
* console.trace() para exibir a *stacktrace* de todos os pontos (ou seja, os arquivos chamados) por onde o código executado passou durante a execução.

No JavaScript, os tipos de dados podem ser divididos em duas categorias: tipos primitivos e tipos de objetos.

Alternativa correta! Muito bem! Os tipos primitivos do JavaScript incluem números, texto (conhecidas como strings) e valores booleanos. Já o tipo objeto é uma coleção de propriedades, onde cada uma possui um nome e um valor, sendo ele um valor primitivo ou outro objeto.

No JavaScript, podemos criar um programa com paradigma funcional e lógico.

Alternativa correta! Certo, podemos sim. O JavaScript é uma linguagem de programação **multiparadigma** e possui suporte para funcional, orientado a objetos ou lógico por exemplo.

Divisão por zero não é um erro no JavaScript.

Alternativa correta! Isso aí. Diferente de outras linguagens de programação como Java ou Python, divisão por zero no JavaScript não gera um erro. O retorno desta operação é infinito ou infinito negativo (Infinity). Porém, a divisão de zero por zero resulta num NaN (*not a number*).

## Nesta aula vimos:

* As características principais do JavaScript, como ser uma linguagem interpretada e dinamicamente tipada.
* O que o NodeJS é um interpretador de JavaScript para backend, ou seja, não executa no navegador, mas sim no servidor.
* Como ler os erros que aparecem no console da aplicação, podendo usar um tradutor, como o Google Tradutor, para entender o que o erro está avisando.
* O que é stacktrace, também chamado de pilha, que nos auxilia a saber o que está acontecendo quando temos um erro, e onde ele se localiza.
* O que é o Console API e como podemos utilizá-lo usando o Console.log() e console.error().
* Onde as mensagem do console devem aparecer dependendo do interpretador que estamos usando, sendo que para o NodeJS as mensagens são mostradas no terminal

OPERADORES DE COMPARAÇÃO

== compara valor

=== compara valor e tipo de dado

* ||: Operador “ou”, retorna true caso uma condição seja válida;
* &&: Operador “e”, retorna true somente se todas as condições forem válidas;
* != e !==: Operadores “não igual” e “estritamente não igual”, utilizados para comparação, da mesma forma que == e === retornam true ou false.

OPERADOR TERNÁRIO

Const idadeMinima = 18;

Const idadeCliente = 16;

Condição true false

Console.log(idadeCliente >= idadeMinima ? “Cerveja” : “Refrigerante”)

TEMPLATE LITERAL / template string

Template é uma tradução de modelo, então, é uma string modelo.

Na template string nós vamos escrever nossa string normal, só que invés de ficarmos concatenando, todos esses mais, na hora que formos colocar o valor da nossa variável, nós vamos colocar o símbolo do cifrão, do símbolo de dólar e abre e fecha chaves.

*const* apresentacao = `Meu nome é ${nome}, tenho ${idade} anos e nasci na cidade de ${cidadeDeNascimento}`;

## Nesta aula vimos:

* A diferença entre fazer comparações com ==, em que o JavaScript faz conversão entre os tipos de variáveis antes de fazer a comparação, e ===, em que tanto o valor quanto o tipo da variável deve ser o mesmo.
* Como é escrito um operador ternário, com o qual fazemos uma comparação entre valores digitando um ?, seguido da possibilidade true, um : e a possibilidade false, ou seja, comparação ? true : false.
* O porquê do nome “operador ternário”, que se deve ao fato de termos 3 operadores juntos em uma única linha para desempenhar uma tarefa e devolver um resultado.
* O uso da template literal, que facilita a construção de strings que demandam concatenação (a operação em que se junta 2 strings), e foi introduzido no ES6.
* Como usar a template literal, escrevendo o texto entre acentos graves () e colocando as variáveis com ${variavel} para que o valor deles fique nessa posição.

FUNÇÕES

A função, ela isola e permite que fechemos um bloco de código (vamos ver como isso acontece) e podemos dizer, exatamente, quando queremos que esse pedaço de código seja executado.

a função tem dois momentos. O primeiro momento (eu vou deixar a minha string na linha 6) da função é quando declaramos a função. Então, declaramos e dizemos o que ela faz. A minha função só imprime texto, mas, a função, também, tem outro momento, que é quando executamos a função uma ou mais vezes.

E como passamos para o código ‘Código, quero que isso aconteça nesse momento’? Nós executamos a função; como executamos a função? Chamando ela pelo nome, então, imprimeTexto e entre parênteses dizemos qual é o texto que queremos que seja impresso.

[06:57] Ju: Lembrando que texto é um nome que damos para sabermos (nós humanos, pessoa que está lidando com o código) o que está sendo esperado. Então, essa função espera receber um texto.

Sobre o que são funções. As funções nesses trechos de código, que separamos em declaração e quando mandamos chamar, a Ju escreveu as duas partes para nós já. Onde tem a palavra chave função, nessa palavra reservada function, é onde está nossa declaração, que no caso está na linha 7, e nós colocamos o nome nela e, em seguida, colocamos o abre e fecha parênteses e dentro botamos o parâmetro.

Depois, fazemos a chamada da nossa função. O interessante da função é que podemos reutilizar uma parte de código, ou seja, conseguimos utilizá-la várias vezes durante o nosso código, durante o nosso programa.

[09:27] Leonardo: E fazer com que nós não precisemos ficar repetindo esse bloco de código várias vezes, escrevendo ele de novo e de novo e de novo.

Esta forma que mostramos é a chamada declaração de função. Ela tem a estrutura que você usa a palavra-chave function, coloca nome, coloca parênteses, abre e fecha chaves. Essa é a forma mais clássica e a forma que existe em várias linguagens bem similares.

s funções ajudam muito no desenvolvimento de um código, pois colaboram para a separação de trechos de código com funções específicas, tornando-o menor e mais legível, O JavaScript nos oferece algumas funções prontas, como é o caso de funções matemáticas (Math em inglês)

PARÂMETROS

Os Parâmetros de Função servem para a função receber informações que são necessárias para ela executar corretamente. Ou seja, para uma função, que soma dois números, funcionar corretamente e funcionar com qualquer conjunto de dois números, que é o que está sendo especificado, nós passamos via parâmetro que ela vai receber num1 e num2.

podemos colocar o nome que quisermos nos parâmetros e eles só vão ser válidos dentro do escopo da função, ou seja, só enquanto a função estiver em execução. Assim que ela acabar a execução estes nomes estão livres para usarmos em qualquer outro lugar. Você pode ter várias funções com os mesmos nomes de parâmetros.

Uma dica importante, é boa prática (não é obrigatória, as funções podem ter quantos parâmetros quisermos, de nenhum até 999, 9999, você escolhe), passar poucos argumentos para a função.

[19:34] Leonardo: Então, se a sua função estiver recebendo muitos argumentos, provavelmente, você consegue quebrar ela em várias funções e fazer várias chamadas menores, várias funções pequenas que vão ficar mais fáceis de entender e vai ficar mais fácil na manutenção deste código depois.

**Parâmetros x argumentos: Na prática se referem ao mesmo tipo de dado; algumas documentações se referem a parâmetros no momento em que a função é definida (no caso, numero1, numero2, etc) e argumentos como os dados que utilizamos para executar a função (ou seja, 30, 45, etc).**

EXPRESSÃO DE FUNÇÃO

Funções e var são listadas no topo do código

Diferença principal : hoisting

ARROW FUNCTION

 Primeiro eu crio uma const, eu vou chamar de apresentarArrow (só para sabermos o que estamos falando). A diferença é: eu coloco um igual (lembrando que criamos uma const, passamos um identificador, um nome para ela, um igual e atribuímos do lado direito do igual), então, o que vamos atribuir?

Primeiro, vamos passar o parâmetro, então, temos um parâmetro para passar nessa função, que é = nome. Em seguida, o que vem? Vem a arrow, a flecha, então, vamos passar a flecha =>. E abro e fecho chaves? Não, não preciso passar chaves, nesse caso, para essa função, porque tudo que essa função tem que fazer é uma linha só de código.

Então, como ficaria? Ficaria const apresentarArrow = nome =>, eu preciso passar, nesse caso, somente o que a função vai retornar, ou seja, uma *string* com a variável ${nome}. Então, a *key* é uma função, uma *arrow* *function* clássica.

[03:07] Ju: Eu tenho só um parâmetro. Eu não tenho, nem parênteses para fechar os parâmetros e nem chaves para fechar o bloco de código do que a função tem que fazer.

Eu vou criar mais uma *const*, vou chamar de const soma (vou enviar a função soma), de novo. Passamos o igual e vamos atribuir.

[03:36] Ju: Como eu tenho dois parâmetros, nesse caso, aí sim eu preciso abrir e fechar parênteses, senão o JavaScript fica meio perdido.

[03:47] Ju: Então, const soma = (num1, num2). Entro com a flecha, a *arrow*, a seta. E preciso do return? Nesse caso, também não preciso do return, porque eu só estou passando uma instrução de uma linha. Qual é a instrução que eu vou passar? É a mesma que já vimos anteriormente, que é só => num1 + num2.

Essa *arrow* *function* veio junto com o ES6, que foi o de 2015. Foi uma grande novidade, como a Ju disse, e ela deixou nosso código bem melhor. Conseguimos, agora, declarar tudo numa linha só, não precisa de chaves, não precisa de *return*, não precisa de nada disso.

[04:42] Leonardo: Ela, também, não pode ser nomeada. Ela não tem nem essa possibilidade para podemos usar, então, fiquem um pouco atentos com isso. E ela sempre vem com uma *const* e o nome da nossa variável, que vamos usar antes.

Se você tiver mais de uma linha, então, você vai precisar usar as chaves e usar o return. É sempre bom ter isso em mente.

## Nesta aula vimos:

* O que são funções, pequenos trechos de código que podem ser executados uma ou mais vezes.
* As vantagens de se usar uma função, pois com elas conseguimos deixar o código mais curto e mais legível, facilitando a manutenção.
* Como retornar informações da função, utilizando o return, lembrando que o console.log() apenas mostra a informação no terminal e não para outras partes do código.
* A utilidade dos argumentos, já que com eles podemos passar variáveis para as funções poderem usar os valores.
* Que com o hoisting o JavaScript analisa todo o código procurando por variáveis declaradas com var e funções para trazer tais declarações para o início do código.
* Expressões de função, uma maneira diferente de montar funções usando variáveis do tipo const e chamando-as pelo nome. Lembrando que é necessário que o programa passe pela variável antes de podermos chamá-la, já que não há suporte à hoisting.
* Arrow function, uma função declarada de maneira mais compacta usando uma const. A arrow function também não tem suporte à hoisting.

**DÚVIDAS**

**Preciso deixar claro que é um ponto flutuante?**

PESQUISAS

**métodos** e **propriedades**