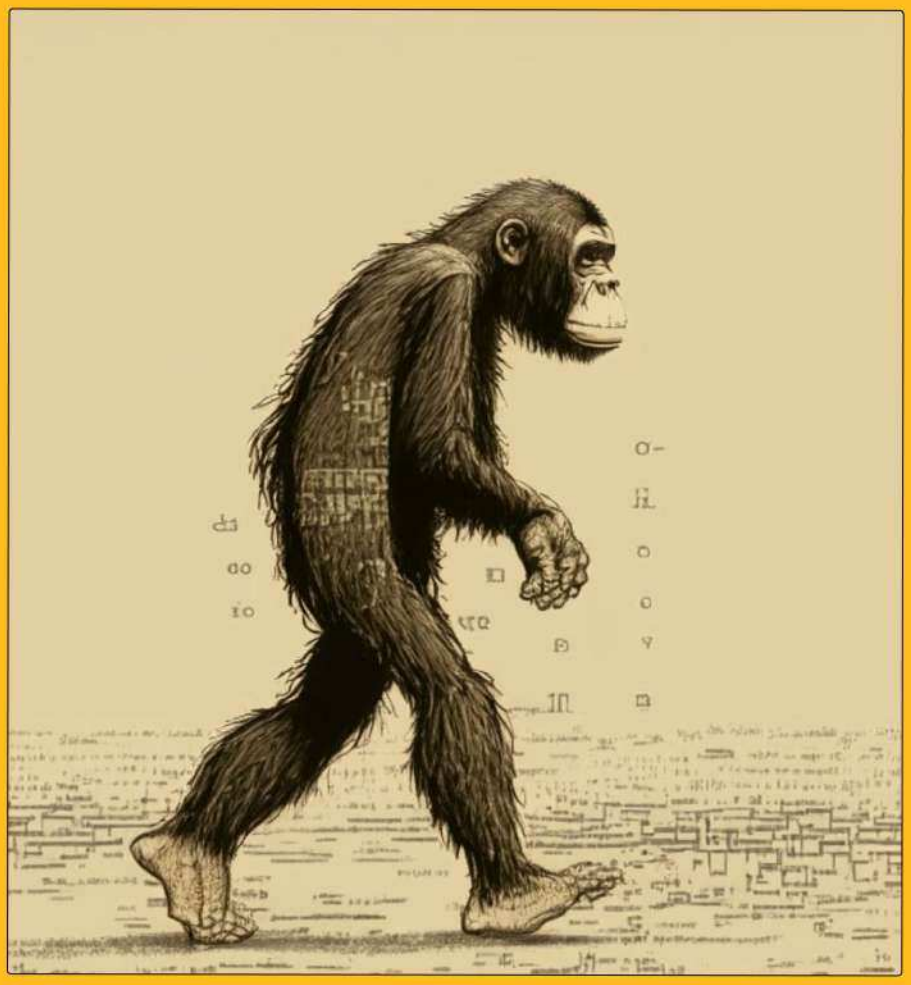
LÓGICA DE PROGRAMAÇÃO GERAL COM ÊNFASE EM JAVASCRIPT



Introdução

A maioria de nós se depara com uma das decisões mais pertinentes e muitas vezes difíceis de nossas vidas: qual carreira seguir. Mediante isso, segue-se outra série de perguntas: o que eu quero deixar para o mundo? O que eu gosto de fazer? Eu tenho talento para isso? Com o trabalho que escolhi, conseguirei o meu sustento ? Muitas vezes, infelizmente, somos levados a negligenciar muitas dessas questões, e ficamos presos apenas no desempenho econômico de determinada atividade, ou imaginamos que não temos talento para isso, etc. Não saber o alcance do potencial que uma determinada atividade tem em si nos retém de explorar e conhecer nossos próprios talentos.

Quando se trata de programação, você vê constantemente milhares de pessoas interessadas no assunto mas limitadas pela própria crença de que é preciso nascer com um talento nato para exercer essa função. Já estou argumentando que isso é um equívoco porque é evidente que o esforço supera a capacidade. E tudo que não sabemos parece muito complicado.

Nunca desistir é um lema com o qual temos que conviver ao longo de nossas vidas, se quisermos alcançar nossos sonhos.

O objetivo deste livro é que você, caro leitor, compreenda um dos principais pilares de qualquer jornada de programação de software. Ele fornece a base para qualquer que seja a linguagem de sua escolha ou qualquer curso de tecnologia pelo qual você seja mais apaixonado. Nós chamamos este pilar de *Lógica de Programação*. Com isso, coordenamos e estruturamos nosso pensamento até a implementação do código, o que nas leva à observação já citada pelo grande Steve Jobs:

“Todo mundo deveria aprender programação de computadores. Porque ensina a pensar."

Com base nessa famosa citação, em cada capítulo veremos o que é lógica de programação, e o como ela se aplica em diversas situações do nosso cotidiano.

Vamos exemplificar e praticar, para que a partir de agora a forma como você enxerga as situações e os objetos ao seu redor fique muito mais organizada e compreensível. E claro, começamos a programar com base nesses conceitos.

Sendo assim, daremos início agora à nossa jornada. Lhe aguardo na próxima página.

O que é lógica de programação?

Pense em qualquer tarefa que você execute ou queira executar no seu dia a dia. Desde a coisa mais simples, como escovar os dentes, até pôr em prática uma receita, ou dirigir. Para executar tais funções, seguimos um passo a passo.

Por exemplo: Para escovar os dentes, pegamos a escova, abrimos a pasta, a colocamos na escova e assim por diante. Em uma receita não é diferente. Seguimos o passo a passo e as medidas correspondentes ao que queremos cozinhar. A lógica de programação é exatamente isso. Um passo a passo, uma receita para se atingir um objetivo final. Esse objetivo é definido conforme o propósito do software a ser desenvolvido. Assim como um bolo de cenoura com calda de chocolate têm um *como* fazer e a hora de cada ingrediente incorporar na receita, o programa a ser desenvolvido também necessita dessa sequência. Desta forma, nós temos segurança de que a massa do nosso bolo vai ficar boa, o tempo certo no forno, para que ao final o sabor esteja o melhor! Fazendo uma analogia, desenvolvendo uma boa lógica de programação, nós desenvolvemos algoritmos mais eficientes, nos torna hábeis para prevenir erros e solucioná-los, caso eles aconteçam. Uma vez que você incorpore os conceitos e regras da lógica de programação, você estará apto a programar em qualquer linguagem, pois todas as linguagens compartilham da mesma lógica.

A sua missão é pensar de forma analítica, encontrar a melhor rota para se chegar ao destino desejado.

Como você já deve ter percebido, qualquer coisa que nós queremos executar em um computador é feito por comandos. Eu peço: “Computador, abra essa janela” , “Desligue”, “Ligue”, “Reinicie” … Para isso nós não falamos o nosso idioma, mas utilizamos as linguagens de programação. Essas linguagens são feitas com os algoritmos, ou seja, o código em si. Veja o exemplo abaixo

Na imagem acima nós estamos dando um comando para o computador, que resultará em uma janela no nosso navegador com a mensagem “Olá,mundo!”

É assim que nós nos comunicamos com um computador: dando instruções baseadas na sua linguagem, de forma que ele execute o que nós pedimos. Mas como nós podemos falar de uma forma que ele nos entenda? Assim como quando falamos com um estrangeiro ou com uma criança, nós precisamos ser específicos para que nos façamos entender, e a nossa mensagem seja compreendida de forma correta. Para fazer isso, vamos compreender como funciona a ‘mente’, ou melhor, a lógica de um computador.

Por trás da mente de um computador

Tenha uma coisa em mente, para nunca mais esquecer: computadores são literais. Ou seja, não entendem conceitos de subjetividade e nem raciocinam para saber lidar com exceções. Por isso, um algoritmo é programado para lidar com diferentes situações, em diferentes contextos. Um computador é exato. Aquilo que nós digitarmos em nosso código é exatamente o que ele irá executar. Nem mais e nem menos.

Como você já deve ter notado, as instruções a serem executadas são passadas através de *algoritmos.*

E afinal, **o que vem a ser um algoritmo?**

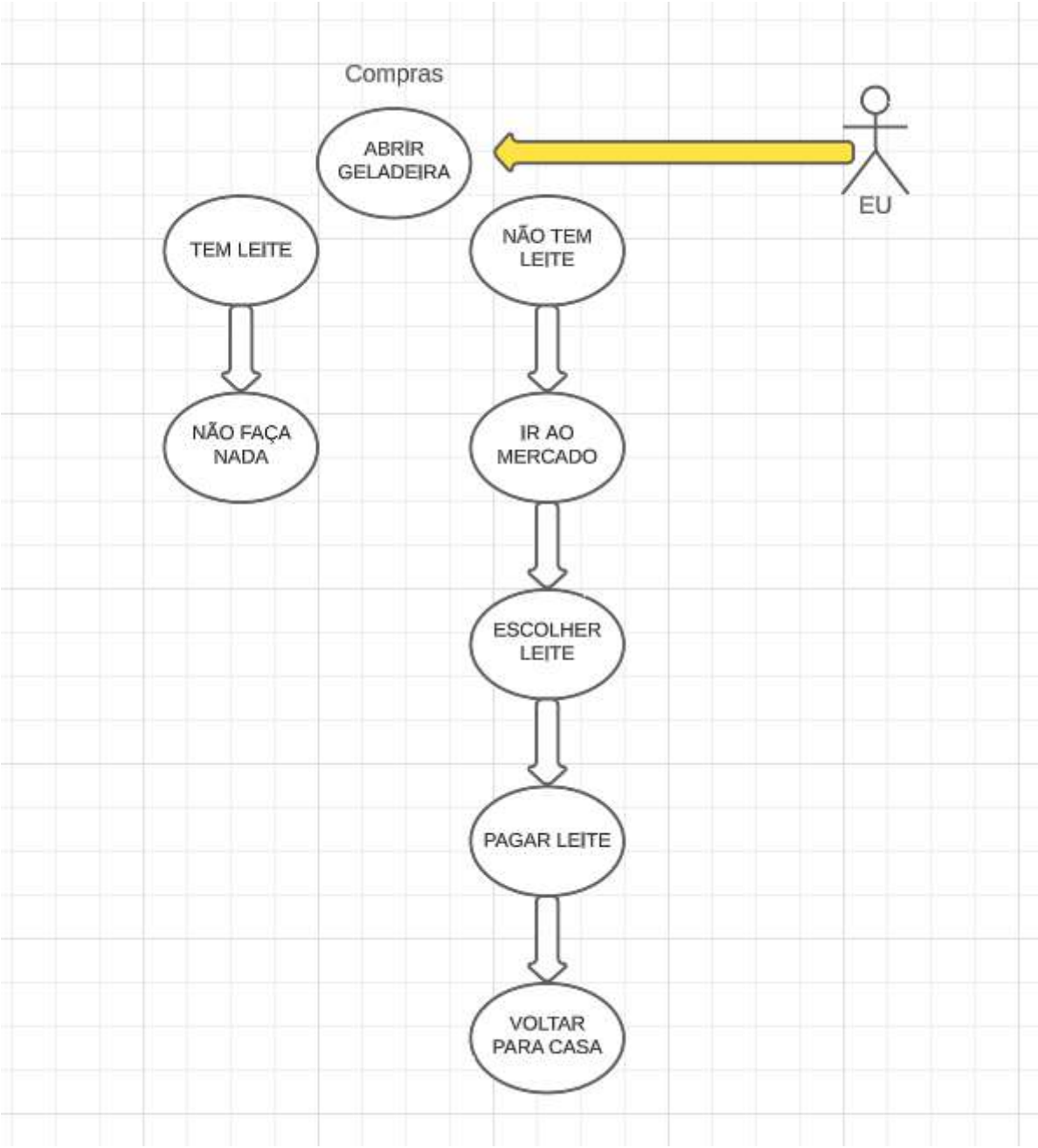
Um algoritmo é um conjunto de passos finitos e organizados que, quando executados, resolvem um determinado problema. É a execução da nossa receita.

A partir de agora nós vamos adentrar no conceito de abstração. Esse conceito nada mais é do que utilizar situações e objetos reais em nossa vida e transformá-los em algoritmo, ou seja, desenvolver um código. Vamos começar fazendo isso com um padrão chamado *UML*.

Vamos ver como isso funciona

O que é UML (Unified Modeling Language)

Observe a figura abaixo com atenção:



Na figura ilustrada acima nós temos a exemplificação do que é um *UML*, também conhecido como **Caso de uso**. É uma forma de ilustrarmos condições e instruções por meio de figuras geométricas.

No nosso exemplo, nós temos um personagem EU, que abre a geladeira e verifica se têm leite. Perceba que se houver, nada é executado. No entanto, caso não haja leite na geladeira, uma cadeia de instruções é programada “ ir ao mercado→escolher leite→pagar leite→voltar para casa”.

Com esse modelo, nós implementamos por meio da linguagem de programação escolhida, ou seja, o idioma que estamos falando, o código que executará essas instruções.

Existem diversas ferramentas online gratuitas que nos permitem fazer tais diagramas UML, deixarei ao final dessa página algumas sugestões.

https://www.lucidchart.com

https://www.miro.com

https://pt.venngage.com/

Praticando UML

Como vimos anteriormente, diagramas de casos de uso são muito úteis e muitas vezes subestimados pelos programadores. Lembro-me de ler um livro chamado “Comece pelo porquê”, onde fala exatamente que antes de começarmos pelo como implementar alguma coisa, devemos desenhar claramente a nossa ideia, ter um porque, uma lógica para o caminho que escolhemos percorrer. Por ser de fácil compreensão, também podemos fazer UML em um papel convencional. Com base nisso, lhe proponho um desafio.

Utilizando algum site para diagramação ou em uma folha de papel, desenhe um caso de uso utilizando as seguintes informações:

Faça de conta que nós queremos cozinhar um purê de batatas para 10 pessoas. Com base nisso, nossa missão é verificar a quantidade de batatas que temos em nossa dispensa. Se tivermos menos de 6 batatas, temos que comprar mais 4, de acordo com a receita. Se tivermos mais de 6 batatas, não precisamos comprar mais.

Algoritimo do purê de batatas.

```
1→ Contagem de batatas

2→ Se batatas >  6 não realizar compra

3→ Se batatas < 6 realizar compra
```

Estrutura dos algoritmos

Agora que nós compreendemos como a lógica se une ao algoritmo e vice-versa, vamos entender mais como funciona a estrutura geral de um algoritmo. A sua sequência, e em que isso resulta para nós.

Em geral, um algoritmo possui três partes. São elas:

Pegando como referência o nosso último exemplo da receita do purê de batatas, temos dois dados de entrada: A quantidade de batatas que a receita pede, e a quantidade que temos na dispensa. Com base nessas duas informações de entrada, vamos para a fase de processamento, a nossa condição de compra ou não de mais batatas, e por último o resultado desse processamento: se fomos ao mercado comprar mais ou não.

Entendido isso, eu proponho agora você a desenvolver o seu primeiro algoritmo. Prepare uma folha de papel, lápis ou caneta e desenvolva o seguinte programa com as informações passadas :

Desenvolvendo o seu primeiro algoritmo

Assim como nós discutimos nas primeiras páginas desse livro, em programação nós utilizamos muito o conceito de abstração. Ou seja, transformar coisas e situações da nossa vida em dados, algoritmos e assim, desenvolver um programa. Pense agora na seguinte situação:

Você está cursando um determinado curso, e precisa calcular a sua média. Implemente essa lógica levando em conta que você tem 4 provas a serem calculadas para uma média final:

Quais são os dados de entrada?

Quais são os processamentos a serem feitos?

Quais são os dados de saída?

RESULTADO DO ALGORITMO CALCULAR MÉDIA

```
p1 = nota1

p2 = nota2

p3 = nota3

p4 = nota4

Média = (p1+p2+p3+p4)/4

Fim do algoritmo.
```

Quais são os dados de entrada?

R: Os dados de entrada são p1,p2,p3,p4

Quais são os processamentos a serem feitos?

R: O processamento está na soma e divisão das notas

Quais são os dados de saída?

R: Fim do algoritmo. (veremos em breve formas de apresentar a saída para o usuário).

Elementos de programação

Até agora nós vimos e exercitamos o que são algoritmos, o que precisamos especificar, e a estrutura para uma implementação de código antes de executá-lo.

De agora em diante nós vamos aprofundar nos elementos de programação de fato. Os blocos que, quando combinados, dão vida a todos os programas que conhecemos e utilizamos. Basicamente todos os softwares, aplicativos e programas que interagimos no nosso dia-a-dia são combinações diferentes, cada um com o seu propósito desses mesmos elementos.

Que elementos são esses?

Nas próximas sessões, nós vamos mergulhar em cada um desses elementos, e então estaremos prontos para desenvolver funcionalidades cada vez mais complexas e eficientes. Partiremos para a prática, entendendo para que e quando é conveniente a implementação de cada um deles. Tendo esse aprendizado bem consolidado, você estará à frente de muitos estudantes e até programadores que possuem formação, mas estagnam na hora de codificar, porque ignoraram os princípios básicos e essenciais para a execução de qualquer sistema.

Siga esses passos, e com certeza a sua jornada na programação será mais prazerosa e consistente.

Vamos adiante!

VARIÁVEIS E CONSTANTES

Você já deve ter ouvido falar que em nosso computador nós temos dois espaços de memória. Um deles é chamado Memória RAM, e o outro é chamado Disco rígido, ou popularmente chamado de HD.

A memória RAM é responsável por armazenar àquelas informações rápidas de curto prazo. Ela é responsável por abrir programas, editar arquivos, realizar operações que estão a serem executadas no momento atual; Já o HD é responsável por guardar dados, arquivos a serem utilizados posteriormente. Nele ficam armazenados jogos, programas, arquivos…

Entendido isso, podemos fazer uma correlação e dizer que as *variáveis* e *constantes* seriam a nossa memória RAM da nossa aplicação, ou seja, do nosso algoritmo. O nosso programa precisa manipular algumas informações no processo de desenvolvimento. Então podemos classificar as variáveis como responsáveis por armazenar dados na memória. Elas são como um rótulo de uma caixa, que dá nome ao conteúdo que se encontra dentro. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor à cada instante.

Vamos elucidar isso com o seguinte modelo:

Quando nós abrimos um GPS para nos auxiliar em nosso trajeto, são requeridas duas informações:

De onde estamos partindo e para onde queremos ir. Essa é uma situação onde nós temos dois campos a serem preenchidos, e cada informação será armazenada em uma variável.

```
destino_1 = “Rio de Janeiro”;

destino_2 = “São Paulo”;
```

se estivéssemos programando, **destino_1** e **destino_2** seriam as nossas variáveis. Elas vão encapsular o conteúdo dos nossos dados.

TIPOS DE DADOS

As variáveis e constantes podem armazenar quatro tipos de dados:

→ Numéricos

→ Caracteres

→ Alfanuméricos

→ Lógicos (tambem chamado de booleano)

IDADE = 30;

NOME = “JOÃO”;

SENHA = “abc123”;

CASADO = SIM;

Agora observe bem os tipos de dados e as variáveis declaradas acima. Você seria capaz de classificar o tipo de cada variável declarada? É interessante que você faça esse exercício e escreva ao lado o tipo de dado de cada variável.

A resposta para isso seria:

```
IDADE é um valor númerico;

NOME é um valor caracter;

SENHA é um valor alfanumérico;

CASADO é um valor lógico.
```

OPERADORES

Os operadores em programação nos permitem realizar cálculos, fazer comparações e manipular valores de variáveis.

Uma vez que nós temos os valores atribuídos às nossas variáveis, nós precisamos manipular essas informações. É aqui que entram os operadores. Eles serão úteis na fase de processamento dos dados.

Utilizando os operadores nós **incrementamos**, **decrementamos**, **comparamos** e **avaliamos** informações dentro do computador. Até mesmo na escola nós temos contato com esse tipo de elemento, quando utilizamos os operadores aritméticos. Você se lembra deles?

Os operadores aritméticos são uma das categorias de operadores em programação. Eles incluem a adição, subtração, multiplicação e divisão.

```
- + adição
- - subtração
- * multiplicação
- / divisão
- ** exponenciação
```

Além dos que já conhecemos, também temos os **operadores relacionais**

Operadores relacionais são utilizados com intuito de comparar dois valores, e sempre retornam um valor booleano **(verdadeiro ou falso).**

Comparações são muito úteis quando queremos tomar uma decisão baseada no valor booleano retornado.

- **==** **(Igual. Verifica se dois valores são iguais. Retorna verdadeiro se forem iguais e falso caso contrário)**
- **!=** **(Diferente. Verifica se dois valores são diferentes. Retorna verdadeiro se forem diferentes e falso caso contrário)**
- **>** **(Maior que. Verifica se um valor é maior do que outro. Retorna verdadeiro se for maior e falso caso contrário)**
- **<** **(Menor que. Verifica se um valor é maior ou igual a outro. Retorna verdadeiro se for maior ou igual e falso caso contrário)**

- **>= (Maior ou igual. Verifica se um valor é menor do que outro. Retorna verdadeiro se for menor e falso caso contrário)**
- **<= (Menor ou igual. Verifica se um valor é menor ou igual a outro. Retorna verdadeiro se for menor ou igual e falso caso contrário)**

Com base nos conhecimentos a respeito de operadores, responda a questão seguinte com V ou F. Consulte a tabela acima, caso seja necessário.

```
idade = 18;

( ) idade == 18

( ) idade != 18

( ) idade > 18

( ) idade < 18

( ) idade ≥= 18

( ) idade ≤= 18
```

Resposta:

V, F, F, F, V, V

Porque

idade é igual a 18

idade não é diferente de 18

idade não é maior do que 18

Idade não é menor do que 18

Idade é maior OU igual a 18

Idade é menor OU igual a 18.

Considerando o valor da variável idade = 18, atribua o valor correto a variável em branco abaixo (de acordo com a lei de trânsito brasileira, idade 18 pode dirigir?) , e classifique respondendo o TIPO de dado que ela retorna.

```
idade = 18

podeDirigir = ‘’;

obs: aspas vazias dentro de uma variável retorna vazia. Ou seja, não tem nada dentro,
mas podemos atribuir um valor para ela quando for necessário.
```

Resposta:

```
podeDirigir = SIM;
```

Retorna um valor booleano.

Operadores lógicos

Vamos finalizar o nosso estudo de operadores falando do último tipo de operador: Operadores lógicos. Nós temos três tipos de operadores lógicos, são eles:

```
- AND (significa E)
- OR (significa OU)
- NOT (significa NÃO)
```

Para um melhor entendimento, leia as frases abaixo:

```
“Eu quero lanchar hambúrguer ou pizza”

“Eu jogo futebol e basquete”

“Se chover eu não vou à praia”
```

Perceba que os operadores lógicos trabalham com alternativas. Em programação nós os utilizamos para uma tomada de decisão baseada no resultado da condição que nós implementamos.

No exercício que nós fizemos anteriormente com a variável idade, por exemplo:

```
idade = 18

podeDirigir = SIM

Poderíamos dizer em uma sentença que João é maior de idade E pode dirigir.

Também poderíamos afirmar que ou João é menor de idade OU João pode dirigir

E também que, se João for menor de idade ele NÃO pode dirigir
```

Baseado nisso nós poderíamos implementar uma série de condições e encadeamentos de código que tratariam essas alternativas. Nós estamos a um passo de entender e praticar esses conceitos utilizando a linguagem JavaScript como modelo, e então vamos fazer funcionar tudo o que temos aprendido até agora.

Estruturas de controle de fluxo

Estruturas de controle de fluxo são responsáveis por tomar uma decisão. Como assim? Nós temos dois caminhos, e essas estruturas são responsáveis por decidir qual caminho seguir. Não se apresse, tudo ficará muito claro…

Tenha você percebido ou não, nós utilizamos esse conceito em praticamente todos os nossos exemplos até aqui, e o praticamos o tempo todo em nosso dia-a-dia.

```
“Se eu tirar 7 na prova, eu passo de ano”

“Se não tiver batata suficiente eu vou ao mercado”

“No Brasil você pode tirar habilitação se tiver 18 anos ou mais”
```

A palavra **SE** aparece em todas as sentenças, e ela determina o que acontecerá em seguida, caso o valor seja SIM ou NÃO. Em uma linha reta de raciocínio, podemos dizer que:

```
Se tirar 7 na prova → passa de ano

Se não tirar 7 na prova → não passa de ano

No Brasil, se você tem 18 anos ou mais → pode tirar habilitação

Se não tem 18 anos ou mais → não pode tirar habilitação
```

É uma forma coesa e sucinta de tomarmos uma decisão com base em um retorno booleano. SIM ou NÃO. Em outras palavras, nós estamos a estabelecer uma **CONDIÇÃO**. Observe você mesmo como nós temos esse pensamento automaticamente em nosso cotidiano.

Estruturas de repetição

As estruturas de repetição são recursos utilizados na programação para automatizar tarefas que exigem a execução de um mesmo bloco de código diversas vezes. Essas estruturas permitem que o mesmo conjunto de instruções seja executado várias vezes sem a necessidade de escrever o código repetidamente, o que otimiza a programação e reduz a possibilidade de erros.

Pense em uma situação trivial, como lavar a louça. Enquanto a louça estiver suja, você segue lavando. Quando não houver mais louça para ser lavada, você pára de lavar. Simples e direto, as estruturas de repetição funcionam com base nessa ideia de executar uma função *se* alguma coisa e até determinado ponto. O algoritmo segue executando uma determinada função enquanto uma condição for verdadeira.

Agora visualize uma garrafa vazia. Você quer enchê-la d'agua. Enquanto a garrafa estiver vazia, você continua enchendo, e quando chegar no limite de água, você pára de encher. Você não precisa tirar e colocar a garrafa, mas sim continuar enchendo até atingir um ponto onde não precise mais continuar fazendo isso.

Enquanto garrafa vazia → encher garrafa

Enquanto garrafa cheia → não encher garrafa

Em programação nós temos 4 estruturas de repetição básicas para praticamente todas as linguagens de programação:

- **While (enquanto).** Repete um bloco de código enquanto uma condição permanecer verdadeira. Caso a condição seja falsa, os comandos dentro do while não serão executados e a execução continuará com os comandos após o while

```
numeroConvidados = 100;

while(numeroConvidados ≤= 100) {

'pode entrar na festa'
```

```
}else{

‘não pode entrar na festa’

}
```

No exemplo acima, vimos uma condição, que se não atendida, ou seja, se não for verdadeira, tomará um outro caminho. Se o número de convidados for menor ou igual a 100, as pessoas poderão entrar. Quando esse condição deixar de ser verdadeira, ou seja, quando o número de convidados passar a ser maior do que 100, não poderá mais entrar gente. Perceba a palavra *else*, que em inglês significa *se não*.

Estruturas de repetição II

- **Do While (faça enquanto).** A ideia é executar o bloco de código pelo menos uma vez e, em seguida, verificar a condição de saída no final do loop. Se a condição for verdadeira, o loop continuará executando. Caso contrário, o loop será interrompido.

```
 condicional = true
 contador = 0

contador = 0;

do {
  console.log("Contador: " + contador);
  contador++;

} while (contador < 5);
```

Neste exemplo, temos um resultado muito parecido com a estrutura While. Mas atenção: a verificação da condicional só é realizada ao final de cada execução do bloco de código, ao invés do começo.

A repetição Do While é utilizada em sistemas onde é preciso executar o bloco de código pelo menos uma vez, necessariamente.

- **For (para).** É utilizado quando se sabe exatamente quantas vezes o bloco de código deverá ser executado. A estrutura é composta por três partes: a inicialização de uma variável, a condição para a execução do bloco de código e o incremento/decremento da variável inicializada.

Por exemplo:

```
for (i = 0; i < 5; i++) {
console.log("O valor de i é: ", i)
}
```

Neste caso, o bloco de código será executado 5 vezes, pois a variável "i" é inicializada com 0 e é incrementada a cada execução até que a condição "i < 5" seja falsa.

A estrutura For é muito utilizada em situações onde é necessário percorrer um vetor ou matriz, por exemplo. Veremos o que são vetorez e matrizes mais à frente.

- **Foreach**

Considerando que a palavra Foreach traduzida para o português significa “para cada”, já podemos ter uma noção da utilidade dessa estrutura de repetição. O foreach é utilizado para quando queremos percorrer por cada elemento do começo ao fim. Quando chegamos ao final da lista, a repetição é terminada.

```
itens = [1, 2, 3, 4, 5]
itens.forEach((valor) => {
  console.log(valor)
})

aqui estamos percorrendo cada número que existe dentro da variável itens. O nome valor
corresponde à cada número dentro da variável itens. Poreríamos nomeá-lo de qualquer coisa.
```

Final da sessão I

Se você chegou até aqui, posso lhe assegurar que a sua base para se tornar um bom programador está formada. Vimos como funciona a estrutura de um algoritmo, operadores, estruturas de repetição e em quais circunstâncias utilizamos cada um deles.

Também vimos que esses conceitos abstratos estão inseridos em atividades que desempenhamos em nossa vida, no mundo real, e fazemos isso de forma automática.

Nas próximas sessões você vai ser inserido nas ferramentas de desenvolvimento, como editor de código e sintaxe da linguagem JavaScript, e tudo o que vimos até agora vai tomar forma, fazendo mais sentido do que antes.

Adianto que teremos muita prática, e por isso garanto que, seguindo cada passo aqui descrito, você terá um conhecimento aprofundado da linguagem.

Prepare-se para um mundo novo e empolgante que está a se abrir a partir de agora.

Vamos compreender o que torna JavaScript uma das linguagens mais utilizadas no desenvolvimento web e destrinchar seus conceitos, funções e efeitos.

Até a próxima sessão!

Por que aprender JavaScript?

O que é JavaScript?

JavaScript é uma linguagem de script de alto nível que é interpretada por navegadores da web. Ele é usado principalmente no lado do cliente, o que significa que é executado no navegador do usuário em vez de em um servidor web. É uma linguagem orientada a objetos, o que significa que os objetos são os blocos de construção básicos da linguagem. JavaScript é usado para criar interatividade em páginas da web, como menus suspensos, janelas pop-up e outros efeitos dinâmicos.

Por que aprender JavaScript?

→ JavaScript é uma das linguagens de programação mais populares do mundo, e é amplamente utilizada no desenvolvimento de aplicativos web.

→ Sua curva de aprendizado é uma das menores, ou seja, é indicada principalmente para quem está começando.

→ JavaScript pode ser utilizado tanto no front-end quanto no back-end, e até mesmo na comunicação com banco de dados. Ou seja, é uma linguagem muito versátil.

→ É uma linguagem muito requisitada no mercado de trabalho. JavaScript está sempre entre as linguagens mais utilizadas no ambiente de desenvolvimento.

Essas são apenas algumas das muitas vantagens e razões do porquê aprender e codificar com JavaScript é sempre interessante para qualquer programador.

Chegou a hora de codar. Alongue os dedos 😉

Preparando o ambiente

Primeiro escolha um editor de código de sua preferência. O editor de código é simplesmente o software onde nós vamos codificar os nossos algoritmos. Hoje existe uma gama de opções que podemos baixar gratuitamente. Ao longo do nosso trabalho, eu vou utilizar o editor Vs Code, por preferência. Mas fique à vontade para pesquisar outros editores, caso queira.

Após fazer o download do programa:

```
Crie um documento HTML (index.html)
Crie um documento JavaScript (script.js)
Linkar o script ao html:

<script src="script.js"></script>
```

Para linkar o JavaScript nós temos duas opções:

→ Linkar entre a tag <head></head>

→ Linkar antes do fechamento da tag </body>

Fique à vontade para escolher uma das opções. No meu caso, estará linkado ao final do arquivo, antes do fechamento do </body>

Feito isso, abra o seu arquivo script.js e vamos começar a declarar nossas variáveis.

ELEMENTOS DE PROGRAMAÇÃO com JavaScript

JavaScript - variáveis

Como nós já estudamos em lógica de programação(lembrando que a lógica é a mesma para todas as linguagens), já sabemos que elas servem para armazenar um tipo de dado. Vamos ver algumas características das variáveis em JS.

Em JavaScript, as variáveis são declaradas usando as palavras-chave '**var**', '**let**' ou '**const**', seguidas pelo nome da variável. Por exemplo:

```
var nome = 'Maria';
let idade = 22;
const brasileira = true;
```

Evitam repetições

```
var preco = 100;
var quantidadeComprada = 10;
var precoTotal = preco * quantidadeComprada ;


Aqui podemos notar que para fazer a operação matemática de multiplicação,
nós não repetimosos valores numéricos,
mas apenas declaramos o nome da variável onde se encontra cada valor.
```

Sem valor

```
var valorTotal;

Podemos declarar uma variável sem atribuir um valor, o que retorna undefined.
Significa que o valor não foi definido.
```

Hosting

```
console.log(cidade);
var cidade= 'Rio de Janeiro';
// Retorna undefined
```

```
var pais= 'Brasil';
console.log(pais);
// Retornar Brasil


O código é lido em ordem de cima para baixo. No primeiro exemplo
tentamos acessar o valor de uma variável que ainda não estava declarada,
pois não havia sido lida ainda.
Já no segundo exemplo tivemos o retorno correto,
pois a variável foi declarada primeiro para depois a acessarmos.
```

Permite mudar o valor atribuído

```
var idade = 30;
var idade = 31;

-> idade agora é 31.

let peso= 60;
let peso= 75;

-> peso agora é 75.

const possuiFilhos = true;
possuiFilhos = false;

-> Retorna um erro porque const não pode ter o valor modificado. A palavra const
vem de constante, o que significa que não pode ser mudado.
```

Algumas considerações importantes:

→ Os nomes podem iniciar com $, _, ou letras. Podem conter números mas não iniciar com eles.

→ Camel case. É comum utilizarmos essa forma de escrita, onde a segunda palavra começa com letra maiúscula. **ex: escreverNome;**

→ Case sensitive. É lido exatamente o que foi e da forma como foi digitado. **Oi** não é lido como a mesma coisa que **oi**.

→ Não utilizar palavras reservadas. Palavras reservadas são as palavras que a linguagem já utiliza em seu programa. Por exemplo, o nome de uma variável não pode ser var. Porque var já está reservada para um objetivo. Você pode conferir as palavras reservadas em : https://www.w3schools.com/js/js_reserved.asp

Praticando variáveis em JS.

1→ Declare uma variável contendo a sua comida preferida

2→ Declare uma variável com o ano em que você nasceu

3→ Declare uma variável chamada pedido sem um valor atribuído a ela

Respostas:

```
1-> var comidaPreferida = 'Hambúrguer';

2-> var anoNascimento = 1982;

3-> var pedido;
```

Tipos de dados JS

Em JavaScript nós temos sete tipos de dados ao total. Alguns deles nós já conhecemos, porém com outro nome. Vamos conhecer agora quais são eles :

```
var nome = 'Ana';  // String
var ano = 2023;// Number
var possuiHabilitacao = true; // boolean
var olhosClaros;// undefined
var simbol = Simbol() //Simbol
var profissao = null; // Null
var novoObjeto = {} // Object
```

Nós podemos utilizar o recurso **typeof** quando queremos descobrir ou retornar o tipo de dado.

```
var nome = 'Ana'
console.log(typeof nome)

//String

Nós podemos somar strings,e dessa forma concatenar, ou seja, juntar palavras. Por exemplo:

var nome = "Jorge";
var sobrenome = 'Oliveira";
var nomeCompleto = nome + ' ' + sobrenome;
// as aspas com espaço servem para não ficar grudado um valor no outro

Também podemos concatenar números com Strings, e ao fazer a leitura, o retorno
dessa variável será sempre uma String.

var gols = 1000;
var frase = 'Pelé fez mais de ' + gols +' gols';
```

Obs: Perceba que quando utilizamos o nome de uma variável, não precisamos de aspas
entre o nome dela, mas apenas no texto que não está atribuído a nenhuma variável.

TEMPLATE STRING

```
var gols = 1000;
var frase = `Pelé fez mais de ${gols} gols`;

A template String deixa o código mais legível e exclui a necessidade do operador + para
concatenar. Basta utilizar crase no lugar das aspas e colocar o nome da variável entre ${}
```

Dica : Todos os textos (Strings) devem ficar entre aspas duplas “” ou simples ‘’ , e os números não são declarados com aspas.

Praticando dados em JS

1→ Declare uma variável contendo uma string

2→ Declare uma variável contendo o seu peso

3→ Declare três variáveis. Uma contendo o seu estado, outra contendo a sua cidade e uma outra contendo o seu país de origem

4→ Verifique o tipo da variável que contém o seu peso

5→ Declare uma variável com a seguinte frase utilizando o seu peso declarado com template String:

‘O meu peso atual é ${};

Resposta:

```
var ola = 'Olá,mundo!';

var peso = 61;

var estado = 'Rio de janeiro';

var cidade ='Niterói';

var pais = 'Brasil';

console.log(typeof peso)

var declararPeso = `Meu peso atual é de ${peso}`;


!! Desafio extra !!
E se, para deixar mais completo nós quiséssemos colocar o kg para representar os kilos
da variável peso?

var declararPeso = `Meu peso atual é de ${peso} kg`;

console.log(declararPeso)
```

Números e operadores - JS

```
var idade = 20;
var altura = 1.70  // o ponto entre os números o define como um número decimal
var dia = 5
```

Operadores aritméticos

Em sessões anteriores nós conferimos os operadores aritméticos, aqueles que nós já conhecemos desde a época escolar. Abaixo temos alguns exemplos de como utilizamos em JS:

```
var subtracao = 200 - 100;
var multiplicaçao = 200 * 2
var soma = 200 + 100
var divisao = 200 /2
var expoente = 10 ** 2
var modulo = 10 % 2
```

Operadores aritméticos com strings

Por exemplo:

```
var soma = '10' + 40; // Aqui estamos concatenando, juntando um valor no outro. Resultando em 1040
var subtracao = '100' - 50; // 50
var multiplicacao = '100' * '2'; // 200
var divisao = 'Olá' / 2; // NaN

ou seja, podemos utilizar operadores artiméticos em uma String, porém o tipo de dado
não será number. Portando nossos métodos para números não poderão ser aplicados.
E caso tenhamos caracteres de palavras tentando
realizar uma operação matemática, retornará isNaN, pois não é possível
fazer nenhum tipo de operação artimética entre palavras e números,
como é o caso da variavel divisao
```

NaN = Not a number

```
var altura = 1.70;
var cm = 'cm';
var altura_completa = altura + cm // '1.70cm;
var altura_divisao = altura_completa / 2 // NaN

Perceba que altura_completa retorna uma String com os dois valores concatenados
por esse motivo, altura_divisao não consegue efetuar uma operação matemática
pois não podemos dividir palavras por números.
```

A ordem das operações matemáticas

É um fundamento básico das operações matemáticas. A ordem é : primeiro é executado multiplicação e divisão, depois soma e subtração.

Caso propositalmente deseje priorizar uma certa conta, devemos a colocar em parênteses para que

essa seja calculada primeiro:

```
var conta = 30 + 10 *2;// 80
var conta_2 = (10 + 5) * 2; // 30
var conta_3 = 20 / 2 * 2;// 20
var conta_4 = 20+ 20* 4 + 10/ 2 //65
```

Operadores unários

Operadores unários são úteis para incremento e decremento de um valor:

```
Quando colocamos o incremento após o valor a ser incrementado, primeiro a saída é
o valor atual, e logo em seguida ele retorna o incremento

var numero_incremento = 10;
console.log(numero_incremento++) // 10
console.log(numero_incremento) // 11

Quando colocamos o incremento antes do valor a ser incrementado, a saída é
o valor já incrementado.

var numero_incremento = 10;
console.log(++numero_incremento) // 11
console.log(numero_incremento) // 11


A mesma lógica e sintaxe se aplicam para o decremento.

var numero_incremento = 10;
console.log(numero_incremento--) // 10
console.log(numero_incremento) // 9
```

```
var numero_incremento = 10;
console.log(--numero_incremento) // 9
console.log(numero_incremento) // 9
```

Operadores unários para transformar Strings em números

Os operadores unários que vimos nos exemplos acima possuem uma outra funcionalidade bastante útil e interessante. Quando declaramos um valor numérico entre aspas, ele não é lido pelo interpretador de código como um number, mas sim como uma string. Nós temos como mudar esse comportamento de uma forma simples: adicionando os operadores + ou - antes de cada valor

Lembrando: funciona apenas com números, pois não é possível transformar uma palavra em valor numérico.

Veja :

```
var palavra = 'Olá, pessoal!';
+palavra // NaN
-palavra // NaN

var ano = '2022'; // repare que está entre aspas, então é uma string
+ano // 28. Agora é retornado o número 28 do objeto tipo number, não string
- ano // - 28

Com essa manipulação, aí sim poderíamos executar uma operação matemática:

console.log(ano + 1) // 2023.
```

Praticando números e operadores

1 **Qual é o resultado expressão abaixo? Por que?**

```
var total = 5+ 5 * 4 / 1 + 20;
```

2 **Incremente a variável abaixo e retorne no console o seu valor já incrementado:**

```
var num = 1;
```

3 **O que vai retornar na expressão abaixo?**

```
var conta = “50” + “50”;
```

4 **Utilize a variável conta acima, e retorne no console o valor 100:**

```

```

5 **Divida o valor total abaixo por 4 e retorne no console :**

```
var total = 400;
var moeda = "R$";
var total_cada =
var resultado =
```

Resultado:

```
1

Multiplicação: 5 * 4 = 20
Divisão: 20 / 1 = 20
Adição: 5 + 20 = 25
Adição final: 25 + 20 = 45

Porque segue a regra da precedência de operadores
```

```
2

var num = 1;
console.log(++num)
```

```
3

5050
```

```
4

var conta = +"50" + +"50";
console.log(conta) // 100
```

```
5

var total = 400;
var moeda = "R$";
var total_cada = 400 / 4
var resultado = total_cada + moeda;

console.log(resultado)
```

Boolean e condicionais JS

No início desse livro nós tratamos sobre os valores booleanos, e vimos que eles têm apenas duas saídas possíveis: SIM ou NÃO. Vamos nos habituar a tratá-los como true e false, pois a linguagem é toda na lingua inglesa, e portanto, devemos implementar desta forma.

```
var luz_acesa= true; // verdadeiro
var luz_apagada = false; // falso
```

Condicionais

Para verificarmos e implementarmos condicionais em javaScript, utilizamos as palavras if, else if, else. Veja o exemplo abaixo:

```
var luz_acesa = true;

if(luz_acesa) {
console.log('Apague a luz');
} else {
console.log('Acenda a luz');
}

// O retorno será 'Apague a luz' e não executará o else.
//A condição pára no momento em que encontra uma condicional que retorne true.
```

Ainda se tratando de condicionais, nós temos mais uma alternativa, que é o else if. Caso o if retorne false, nós podemos implementar mais alternativas ao código:

```
var numero = 10

if(numero < 10) {
console.log('O número é menor que 10');
} else if(numero === 10) {
console.log('O número é 10');
}else{
```

```
console.log('O número é maior que 10');


Neste exemplo o retorno será ('O número é 10'),
pois a condição verdadeira é que o número é igual a 10.
Assim o código pára e não executará o else.
```

Switch

O Switch é mais uma alternativa de comparação. Nele nós não podemos utilizar operadores lógicos de comparação, mas se torna uma boa opção para quando queremos verificar um valor e retornar a alternativa true que a corresponde. Caso ela seja igual, nós podemos tomar uma ação e utilizar a palavra chave break; para cancelar a continuação. O valor de default ocorrerá caso nenhuma
das anteriores seja verdadeira

```
var medida= 'M';

switch (medida) {
case 'P':
console.log('Tamanho pequeno');
break;

case 'M':
console.log('Tamanho médio');
break;

case 'G':
console.log('Tamanho grande');
break;

default:
console.log('Nenhum tamanho disponível');



O switch ira percorrer cada caso até encontrar algum que corresponda ao valor de medida.
Caso não encontrasse nenhuma alternativa true, ele cairia no default,
executando a mensagem 'Nenhum tamanho disponível'
```

Operador lógico de negação

Utilizamos esse operador para inverter o retorno do valor booleano. Ao utilizá-lo, o true retorna false e o false retorna true:

```
var fruta = 'Manga';

if (fruta !== fruta) {
  console.log('Não é manga');
}

Ou seja, se o valor de fruta não for manga, retorna true para essa negaçao.
```

Comparação estrita x Comparação não estrita

Esses tipos de comparações levam ou não em consideração estritamente *o que* e *como* está escrito determinada coisa. Quando digitamos **‘nome’** e **‘Nome’** é a mesma palavra, porém o javascript é por sua natureza *case sensitive*, ou seja, leva em consideração a forma como foi escrito. Portanto ‘nome’ com n minúsculo e ‘Nome’ com N maiúsclo são coisas diferentes. Caso queiramos apenas conferir se o valor é o mesmo , sem levar em consideração a forma como foi escrito, podemos utilizar `==` ao invés de `===`

Observe quais utilizam apenas dois sinais de `==` e quais utilizam três sinais de `===` e o que retornam:

```
100 == '100'; // true, pois são o mesmo valor
100 == 100; // true, pois são o mesmo valor
10 === '10'; // false, pois são o mesmo valor mas um é number e outro string
10 != 15 // true, porque 10 é diferente de 15
10 != '10' // false, porque 10 tem o mesmo valor de '10'
10 !== '10' // true, porque 10 não é a mesma coisa que '10'
```

Operadores lógicos `&` (e) `||` ('ou)

Utilizamos esses operadores para realizar comparações e condições.

Por exemplo:

```
var idade = 20

idade >= 20 && idade < 50 // true, pois idade é igual a 20 E menor que 50

com o &&, para retornar true as duas informações devem ser verdadeiras.
```

|| (ou):

```
var nome = 'João'

nome == 'Lucas' || nome == 'João' // retorna true e o valor verdadeiro, nesse caso, 'João'


Diferente do operador &&,  o operador || não precisa que necessariamente
todas alternativas estejam corretas,
mas sim alguma delas.
```

Exercitando Boolean e condicionais JS

1. Faça condicionais com as variáveis abaixo, verificando se o valor em conta é maior ou igual ao valor da compra, e imprima no console ‘saldo maior do que valor da compra’, ‘saldo igual ao valor da compra’, ‘saldo menor que o valor da compra’. E ao final, diga qual resultado será imprimido.

```
var valor_em_conta = 1000
var valor_da_compra = 1000
```

2. Com o operador de negação, inverta o valor atribuído a variável habilitado, e imprima no console ‘pode dirigir’, caso seja true, e ‘não pode dirigir’, caso seja false.

```
var habilitado = true;
```

3. O que aparecerá no console? Por que?

```
if(('Bicicleta' === 'bicicleta') && (5 > 2)) {
console.log('Todas as alternativas são verdadeiras');
} else {
console.log('Uma ou mais alternativas são falsas');
}
```

4. Programe você mesmo uma condicional utilizando o **switch**, que siga a seguinte premissa: Se a cor do sinal for verde, imprima “Siga’ , se for amarelo imprima ‘Atenção!’, se for vermelho imprima ‘pare’ , e caso a variavel cor do sinal for vazia, imprima ‘Sinal quebrado’, e responda qual case será impresso no console.

```

```

Resultado:

```
1.

var valor_em_conta = 1000
var valor_da_compra = 1000
```

```
if( valor_em_conta > valor_da_compra) {
  console.log('Saldo maior do que a compra')
} else if( valor_em_conta === valor_da_compra) {
  console.log('Saldo igual ao valor da compra')
}else {
  console.log('Saldo menor do que o valor da compra')
}

// o resultado cairá no else if 'Saldo igual ao valor da compra'
```

```
2.

var habilitado = true

if(!habilitado) {
  console.log('Não pode dirigir')
} else {
   console.log('Pode dirigir')
}
```

```
3.

if(('Bicicleta' === 'bicicleta') && (5 > 2)) {
console.log('Todas as alternativas são verdadeiras');
} else {
console.log('Uma ou mais alternativas são falsas');
}

//o código cairá no else, pois embora 5 seja maior que 2, 'Bicicleta' não é estritamente igual a 'bicicleta'
```

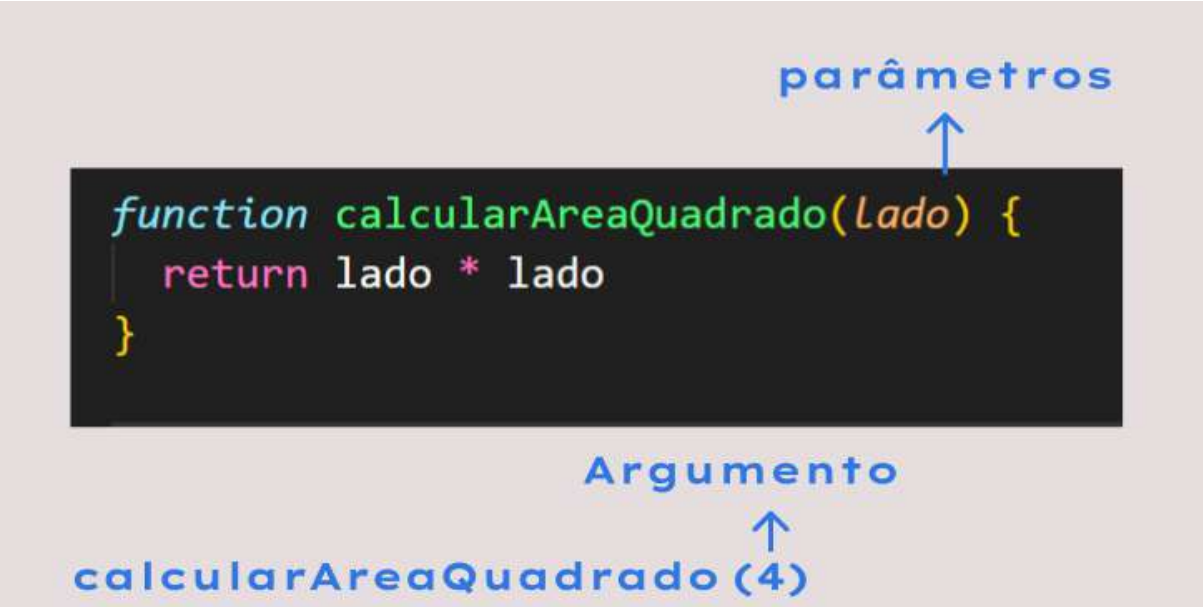
```
4.

var cor_do_sinal = 'Vermelho';

switch (cor_do_sinal) {
  case 'Verde':
    console.log('Siga');
    break;
  case 'Amarelo':
    console.log('Atenção!')
    break;
  case 'Vermelho':
    console.log('Pare')
    break;
  default:
    console.log('Sinal quebrado')
}

// o código imprimirá 'Pare' , pois é a condição que corresponde a variável cor_do_sinal.
```

Funções em JS

Chegamos agora em uma etapa muito importante no estudo de programação em geral: funções.

Como o próprio nome nos intui, as funções são utilizadas quando queremos implementar alguma funcionalidade ao nosso código. Bom, isso nós já estávamos fazendo, mas o nosso código estava jogado, não tinha um pai, não o colocamos dentro de uma caixa para protegêlo e, principalmente, quando for preciso repetir aquele mesmo código, ou parte dele, nós teríamos que digitar tudo de novo, tornando o nosso almoritmo muito verboso e não otimizado. Ou contrário disso, as funções permitem a reutilização de um bloco de código, e a mesma lógica dentro dela pode ser invocada a qualquer momento. Algumas caracteristicas iniciais sobre funções:

1. É um bloco de código que pode ser executado e reutilizado
2. Podemos passar valores esperados para uma função, o que chamamos de parãmetros



Para codificar uma função, primeiro declaramos a palavra *function,* e depois o nome dela. O uso de parênteses é obrigatório, mas os parâmetros são opcionais. Nesse

caso, o parâmetro esperado é lado, e na função nós já dizemos o que deve acontecer. Ao chamarmos a função, o argumento passado é 4. Ou seja, lado * lado = 4 *4. Os argumentos são o valor do que passamos no parâmetro.

continuando funções js

Parâmetros e argumentos

Ao **criar** uma função, nós podemos **opcionalmente** definir parâmetros

Ao **executar** uma função, nós podemos passar argumentos

```
function nomeEidade(nome, idade) {
  var dados_pessoais = nome + idade;
return dados_pessoais;
}

nomeEidade('Carla', 22) // Carla e 22 são argumantos

Podemos ter mais de 1 argumento em uma função, e os separamos por vírgula.
O parâmetro é opcional, ou seja, podemos declarar uma função sem nenhum parâmetro
```

Chamar o nome da função junto ao parênteses executa a função. Se não a chamarmos ela não será executada.

```
function sucoEscolhido(suco) {
if(suco === 'laranja') {
return 'Você gosta de suco de laranja';

} else if(suco === 'abacaxi') {
return 'Você gosta de suco de abacaxi';

} else if(suco === 'maracujá') {
return 'Você gosta de suco de maracujá';

else {
return 'Você não gosta de suco nenhum';
}
}

 sucoEscolhido(); // retorna 'Você não gosta de suco nenhum', pois definimos o parâmetro, mas
não colocamos argumento
```

As funções também podem receber outras funções como parâmetros. É chamado de função de callback. Geralmente essas funções ocorrem após algum evento. Aqui está um exemplo:

```
addEventListener('click', function() {
console.log('Clicou');
});

// A função possui dois argumentos
// Primeiro é a string 'click'
// A segunda é é uma função anônima
```

As funções podem ou não retornar um valor. Quando não denimos o *return*, ela irá retornar *undefined* . O código interno da função é executado normalmente, independente
de existir valor de return ou não.

```
function imc(peso, altura) {
const imc = peso / (altura ** 2);
console.log(imc);
}
imc(80, 1.80); // retorna o imc
console.log(imc(80, 1.80)); // retorna o imc e undefined
```

Escopo léxico

Funções conseguem acessar variáveis que foram criadas no contexto **pai**

```
var profissao = 'Cientista de dados'; // variável global, disponivel para o código todo

function dados() {
var nome = 'Lucas';
var idade = 23;

function outrosDados() {
var endereco = 'São Paulo';
var idade = 30;
return `${nome}, ${idade}, ${endereco}, ${profissao}`;
}
return outrosDados();
}
dados(); // Retorna 'Lucas, 23, São Paulo, Cientista de dados'
outrosDados(); // retorna um erro
```

Hoisting

Antes de executar uma função, o JS 'move' todas as funções declaradas para a memória

```
imc(80, 1.80); // imc aparece no console

function imc(peso, altura) {
const imc = peso / (altura ** 2);
console.log(imc);
}
```

Praticando funções JS

1. Escreva uma função chamada `saudacao` que recebe um nome como parâmetro e retorna a string "Olá, {nome}!".

```

```

2. Crie uma função que verifica se um número é par

```

```

3.Crie uma função que retorne o tipo de dado do argumento passado nela (typeof)

```

```

4. Escreva uma função chamada `soma` que recebe dois números como parâmetros e retorna a soma deles.

```

```

5. Crie uma função que recebe o evento de click e uma função de callback como parâmetro, onde ao clicar apareça o seu nome todo

```

```

6. Corrija o erro abaixo:

```
var totalPaises = 193;

function precisoVisitar(paisesVisitados) {
  return `Ainda faltam ${totalPaises - paisesVisitados} países para visitar`;
}
function jaVisitei(paisesVisitados) {
  return `Já visitei ${paisesVisitados} do total de ${totalPaises} países`;
}

precisoVisitar(10);
```

Resultado:

```
1.

function saudacao(nome) {
  return `Olá, ${nome}!`;
}
```

```
2.

function numeroPar(num) {
  if (num % 2 === 0) {
    return 'É par'
  }
  else {
    return 'É ímpar'
  }
}

OBS! estamos apenas retornando uma operação, não passando um numero como argumento
```

```
3.

function tipoDeDado(dado) {
  return typeof dado;
}
```

4.

```
function soma(num1, num2) {
  return num1 + num2;
}
```

!!Desafio!!
Como você faria para imprimir no console o resultado dessa operação?

R: console.log soma(10, 20)

5.

```
addEventListener('click', function () {
  console.log('Ana Beatriz Oliveira De Paula')
})
```

6.

```
var totalPaises = 193;

function precisoVisitar(paisesVisitados) {
  return `Ainda faltam ${totalPaises - paisesVisitados} países para visitar`;
}
function jaVisitei(paisesVisitados) {
  return `Já visitei ${paisesVisitados} do total de ${totalPaises} países`;
}

precisoVisitar(10);
jaVisitei(10)
```

O erro ocorria pelo fato de ter chamado apenas a função precisoVisitar,
porém estava faltando chamar e atribuir um argumento à função jaVisitei.

Objetos em JS

Os objetos são entidades fundamentais quando se trata de JavaScript. Aliás, um dos conceitos centrais da linguagem é de que ‘tudo é objeto’ , pois tudo contém propriedades e métodos.

Aqui estão algumas informações importantes a respeito de objetos:

- São dinâmicos. Isso significa que você pode adicionar, modificar ou remover propriedades de um objeto, mesmo após ele ter sido criado.
- Propriedades e métodos consistem em chave e valor.
- Acessamos uma propriedade e/ou método por meio do **.** seguido do nome do que queremos consultar

```
var pessoa = {
nome: 'João',
idade: 21,
profissao: 'Estudante',
maiorDeIdade: true,
}
pessoa.nome; // 'João'
pessoa.maiorDeIdade; // true
```

Métodos

Os métodos em JavaScript são semelhantes às funções, mas estão vinculados a um objeto específico. Essas funções podem ser chamadas para executar ações ou manipular os dados do objeto em que estão definidas.

```
var calculadora = {
  somar: function(a, b) {
    return a + b;
  },
  subtrair: function(a, b) {
    return a - b;
  }
};
```

```
var resultadoSoma = calculadora.somar(5, 3);
console.log(resultadoSoma);  // Saída: 8

var resultadoSubtracao = calculadora.subtrair(10, 4);
console.log(resultadoSubtracao);  // Saída: 6
```

Forma abreviada de incorporar os métodos:

Ocultamos a palavra function. Resultado: somar(), subtrair():

```
var calculadora = {
  somar(a, b) {
    return a + b;
  },
  subtrair(a, b) {
    return a - b;
  }
};

var resultadoSoma = calculadora.somar(5, 3);
console.log(resultadoSoma);  // Saída: 8

var resultadoSubtracao = calculadora.subtrair(10, 4);
console.log(resultadoSubtracao);  // Saída: 6
```

Como criar um objeto em JS?

- Um objeto é criado utilizando as chaves {}

```
var produto= {};
var pessoa = {};

console.log(typeof produto); // 'object'
```

- Acesse propriedades de um objeto utilizando o ponto

```
var carro= {
marca: 'Fiat',
cor: 'Branco',
ano: 2022,
}
var carro_marca = carro.marca; // 'Fiat'
```

- Podemos modificar o valor de uma propriedade utilizando o **.** e **=**

```
var carro= {
marca: 'Fiat',
cor: 'Branco',
ano: 2022,
}
 carro.marca = 'Renault'
console.log(carro.marca) //  'Renault'
```

- Podemos adicionar propriedades e métodos mesmo após já ter fechado o objeto. Basta adicionar um novo nome e denir o valor.

```
var endereco= {
cidade: 'Ribeirão Preto',
rua : 'Avenida Ribeirao',
}
endereco.numero= 50;
endereco.estado= 'São Paulo';
```

- Para ativar um método, nós temos que invocá-lo, assim como fazemos em todas as funções.

```
var pessoa= {
nome: 'Ana Beatriz',
idade: 29
saudacao() {
return 'Olá!';
}
}
```

```
pessoa.saudacao();
// 'Olá!'
```

- Uso da palavra **this**

A palavra this faz referência ao próprio objeto declarado.

```
var height = 120;

var menu = {
width: 800,
height: 50,
metadeHeight() {
return this.height / 2;
}
}
menu.metadeHeight(); // 25
// sem o this, seria 60 pois pegaria a variável global, que está antes do objeto
```

Praticando com objetos em JS

1. **Crie um objeto que contenha o seu nome, a sua idade e a cidade onde mora**

```

```

2. Modifique o valor da propriedade preço para R$100.000

```
var carro = {

preco: 85.000,

portas: 4,

marca: 'BMW',

}
```

3. Crie um objeto que contenha as seguintes propriedades: marca de uma televisão, polegadas, se é smartTv ou não e inclua um método de ligar, caso essa função seja ativada.

```

```

4. Inclua a propriedade ‘sexo’ no objeto abaixo, e atribua um valor a ela.

```
var usuario= {

email: 'usuario@email.com',

idade: 32,

estado: 'Paraná',

}
```

Resultado:

```
1.

var pessoa= {

nome: 'Ana Beatriz',

idade: 29,

cidade: 'Rio de Janeiro',

}
```

```
2.var carro = {

preco: 85.000,

portas: 4,

marca: 'BMW',

ligar(on) {

if(on === true) {
      return 'Tv ligada';
    } else {
      return 'Tv desligada';
    }
  }

}

carro.preco = 100.000
```

```
3.

var tv= {

marca: 'Samsung',
```

```
polegadas: 50,

smartTv: true,

ligar(on) {
    if(on === true) {
      return 'Ligada';
    } else {
      return 'Desligada';
    }
  }

}

tv.ligar(true); // Saída: 'Ligada'
tv.ligar(false); // Saída: 'Desligada'
```

```
4.

var usuario= {

email: 'usuario@email.com',

idade: 32,

estado: 'Paraná',

};

usuario.sexo = 'Masculino'
```

Tudo é objeto JS

Em javaScript tudo é ou pode se comportar como um objeto. Ao afimar isso nós estamos querendo dizer que Strings, booleans, números, objetos e etc possuem propriedades e métodos, por isso são objetos.

Obviamente nós não decoramos todos os métodos e propriedades referentes a um determinado tipo de dado. Com a prática nós vamos decorando instintivamente àqueles que temos mais costume de aplicar, porém é muito recomendado olhar a documentação e conferir os métodos e propriedades de determinado tipo de objeto.

Também vale elucidar aqui que à isso se aplica uma das maiores praticas de programadores em geral, que é consultar a documentação.

Vamos ver como isso funciona na prática.

```
var nome = 'Lucas';

nome.length; // 5
nome.charAt(1); // 'u'
nome.replace('as', 'ia'); // 'Lucia'
nome; // 'Lucas'

aqui estamos implementando métodos do construtor String()
```

Number

```
var numeros = [10, 50, 100, 20];
var maior_numero = Math.max(...numeros);

console.log(maior_numero); // Saída: 100

var numero = -10
var numero_positivo = Math.abs(numero)

console.log(numero_positivo ); // Saída: 10

var numero_decimal = 60.5
var numero_inteiro = parseInt(numero_decimal)
console.log(numero_inteiro ); // Saída: 60
```

Sugiro que você dê uma olhada na documentação, para que assim crie o hábito de buscar a melhor opção de método para o que deseja manipular. Lembre-se: Se você está trabalhando com uma string, busca por métodos de string; se estiver trabalhando com um number, busque por métodos de number, e faça isso para qualquer tipo de objeto que deseje manipular.

No site do mozila nós podemos encontrar uma documentação bem completa. Deixo aqui abaixo um link para nos auxiliar nos próximos exercícios:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/String

Praticando objetos e métodos

1. Busque por um método que modifique a string abaixo colocando-a toda em letras maiúsculas:

```
var frase = "Eu adoro programar em javascript"
```

2. Busque e nomeie abaixo 4 métodos para manipular objetos do tipo number (explique para quê serve cada um deles):

```

```

3. Busque e escolha um método para acessar o DOM, e explique o que esse método faz:

```

```

Resultado:

```
1.

var frase = "Eu adoro programar em javascript"
var frase_maiuscula = var frase_maiuscula = frase.toUpperCase();

console.log(frase_maiuscula); // Saída: EU ADORO PROGRAMAR EM JAVASCRIPT
```

```
2.

- parseInt(): Converte uma string em um número inteiro.Por exemplo: parseInt("10") retorna 10.
- isNaN(): Verifica se um valor é NaN (Not a Number). Retorna true se o valor não for um número válido. Por exemplo: isNaN("abc") reto
- toFixed(): Formata um número com uma quantidade específica de casas decimais e retorna uma string. Por exemplo: var numero = 3.14159
-Math.abs(): Retorna o valor absoluto de um número. Por exemplo: Math.abs(-10) retorna 10.
```

```
3.

document.getElementById()
//é um método do objeto document em JavaScript
// usado para obter uma referência a um elemento HTML específico na página
//com base no valor do atributo id desse elemento.
```

Arrays e loops JS

Em programação, um array é uma lista de dados que permite armazenar um conjunto de valores relacionados. São frequentemente usados para armazenar números, strings, objetos. Cada valor de um array éidentificado por um índice. Eles são uma ferramenta poderosa que pode ser usada para armazenar e acessar grandes quantidades de dados de forma eficiente.

Vamos ver como nós declaramos um array e sua sintaxe:

```
var frutas = ['Banana', 'Maçã', 'Abacaxi'];
 frutas[2] // Abacaxi
 frutas[0] // Banana;

os índices, ou seja, a posição de um determinado valor é acessado pelos colchetes.
Lembrando que a contagem começa com 0. [];
```

Como foi dito, temos propriedades e métodos, pois tudo é objeto.

Métodos e propriedades de um array:

```
var frutas = ['Banana', 'Maçã', 'Abacaxi'];
frutas.pop(); // Remove o último item e retorna ele. Saída: Abacaxi
frutas.lenght // Conta quantos elementos tem dentro do array. Saída: 3
frutas.push('Uva') // Adiciona um elemento ao final do Array.
//Saída:  ['Banana', 'Maçã', 'Abacaxi', 'Uva']
```

For loop:

For loop servem para que se repita um código até que uma determinada condição seja atingida.

É dividido em três partes: **início, condição, incremento**

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

```
O que estamos fazendo aqui é iniciando uma variável i com o valor 0,
implementando a condição se i menor que 10
incrementando i até que essa condição seja verdadeira

//Saída: 0 1 2 3 4 5 6 7 8 9 10
```

For loop com arrays:

O loop irá parar caso encontro e palavra break

```
var videoGames = ['Switch', 'PS4', 'XBox', '3DS'];
for (var i = 0; i < videoGames.length; i++) {
console.log(videoGames[i]);
if(videoGames[i] === 'PS4') {
break;
}
}
```

ForEach:

O forEach é um método responsável por executar uma função para cada item da array. É uma forma simples de utilizar loop em arrays.

```
const celulares = ['Samsung', 'Apple', 'Xiomi']
celulares.forEach( function(cel) {
  cel.toUpperCase()
  console.log(cel)
    })

cel corresponde à cada elemento dentro de celulares. Recebe uma função que
coloca todos os cel em letra maiúscula.
```

Praticando Arrays e loops JS

1. Crie um array com números de 1 à 10.

```

```

2. Imprima os números do array utilizando forEach

```

```

3. Pesquise métodos para encontrar o maior e o menor número da array, e os imprima no console

```

```

4. Interaja com um loop nas marcas abaixo e pare ao chegar em Toyota

```
var marcas_carro = ['Chevrolet', 'Ferrari', 'Toyota', 'Fiat', 'Ranault']
```

5. Adicione a marca ‘Jaguar’ no início da lista acima, e retorne o array no console

```

```

Resultado

```
1.

var numeros = [1,2,3,4,5,6,7,8,9,10]
```

```
2.

var numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
numeros.forEach(function (num) {
  console.log(num)
})
```

```
3.

var numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

var maior = Math.max(numeros);
var menor = Math.min(numeros);

console.log(maior);
console.log(menor);
```

```
4.

var marcas_carro = ['Chevrolet', 'Ferrari', 'Toyota', 'Fiat', 'Ranault']

for (var m = 0; m < marcas_carro.length; m++) {
  console.log(marcas_carro[m])
  if (marcas_carro[m] === 'Toyota') {
    break;
  }
}
```

```
5.

var marcas_carro = ['Chevrolet', 'Ferrari', 'Toyota', 'Fiat', 'Ranault']

marcas_carro.unshift('Jaguar')
console.log(marcas_carro)
```

Atribuição e ternário JS

Os **operadores de atribuição** serve como uma forma de abreviação, tornando o código mais limpo. Possivelmente você irá se deparar com muitos códigos escritos desta forma, ao longo da sua jornada:

```
var x = 5;
var y = 10;
x += y; // x = x + y (15)
x -= y; // x = x - y (-5)
x *= y; // x = x * y (50)
x /= y; // x = x / y (0.5)
x %= y; // x = x % y (0)
x **= y; // x = x ** y (9765625)

nós atribuímos a operação de x com y, porém não é necessário repetir o X, mas apenas colocar o sinal da operação a ser realizada junto
```

Operador ternário

É uma boa prática, para abreviação de if e else, quando temos uma condição mais simples e curta:

```
var idade = 19;
var podeBeber = (idade >= 18) ? 'Pode beber' : 'Não pode beber';
console.log(podeBeber) //


o sinal de ? retorna 'Pode beber' se o resultado da condição for true
e o : retorna o resultado, caso o resultado da operação seja false.
```

Pratique:

Como você realiza uma soma e uma multiplicação com os valores abaixo,utilizando operadores de atribuição?

```
num1 = 10
num2 = 20
```

Atribua true para a variável pode_entrar_boate, caso o cliente seja maior de idade e possua identidade, e false caso o contrário:

```
var maior_de_idade = true;
var possui_identidade = true;
var pode_entrar_boate;
```

Resultado:

```
num1 = 10
num2 = 20

num1+=num2
num1*=num2
```

```
var pode_entrar_boate = maior_de_idade && possui_identidade;
```

Fim da sessão

Se você chegou até aqui, parabéns! Espero que ao longo da leitura você tenha compreendido, e, principalmente, praticado os nossos exercícios. O ato de programar é uma arte que vai se aprimorando com a prática.

Para nos tornarmos cada vez mais flúidos em uma linguagem, precisamos primeiro entender os seus fundamentos e sintaxes, além da lógica de programação, que é essencial para qualquer que seja a linguagem escolhida.

Nós fizemos um bom trabalho até aqui, e esteja certo de que teremos muito mais!

Agora que entendemos os pilares, conceitos e sintaxe, eu sugiro que você baixe o e-book de projetos em JavaScript. Ele é gratuito e está disponível para você desenvolver projetos em javaScript, tornando tudo o que aprendeu mais palpável,aplicando o que aprendemos.

Se você deseja programar projetos reais, solicite o seu e-book gratuito pelo email:

suzana.dev@gmail.com

Até breve, dev!

suzana f.