# Lógica e Algoritmos

Público alvo: Iniciantes que queiram aprender sobre lógica de programação e algoritmos.

Pré-requisitos: Informática Básica.

# Índice

# Copyright

## Equipe

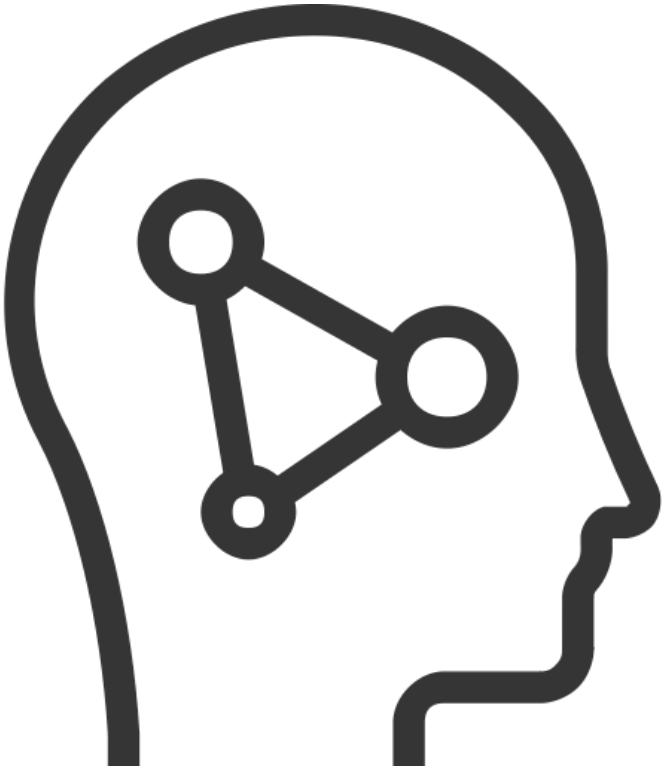
| Conteúdos | Diagramação | Revisão |
|---|---|---|
| Ralf S. de Lima | Fernanda Pereira | Fernanda Pereira |

## Histórico de edições

| Edição | Idioma | Edição |
|---|---|---|
| 1ª | Português | Janeiro de 2018 |
| 2ª | Português | Julho de 2020 |

# Capítulo 01 - Lógica e Algoritmos

Esse capítulo irá abordar os conceitos básicos necessários para compreendermos o uso da lógica e algoritmos no dia a dia de um desenvolvedor, além de sua importância na elaboração de projetos. Com essa base você conseguirá ter os fundamentos necessários para trabalhar com qualquer linguagem de desenvolvimento.



## Compreendendo o uso da lógica

Quando mencionamos o termo lógica, estamos dizendo que alguma ação, função, estudo ou acontecimento possui uma ordem que faz sentido. Na programação esse conceito está ligado ao desenvolvimento de algum software que pode auxiliar uma determinada pessoa ou empresa.

Vamos supor que você tenha várias músicas em seu computador ou smartphone, porém precisa de algo para poder organizar, podendo ser por artista, gênero, tempo, ano de lançamento, entre outras características. Note que temos um problema, que é a falta de organização das músicas, como podemos resolver? Através de um raciocínio lógico teremos que fazer a filtragem de músicas, e em seguida dispor em uma tela para o nosso cliente.

Teremos que ter um fluxo de trabalho com início, meio e fim, na programação precisamos sempre garantir que tenha sido executado um determinado passo para que o próximo consiga ser executado com perfeição. No nosso exemplo de organizar as músicas precisamos primeiro saber quais arquivos de áudio estão no dispositivo de nosso cliente, depois fazer a filtragem, se tentarmos fazer a filtragem antes de sabermos quais arquivos de áudio o dispositivo possui não irá funcionar, correto?

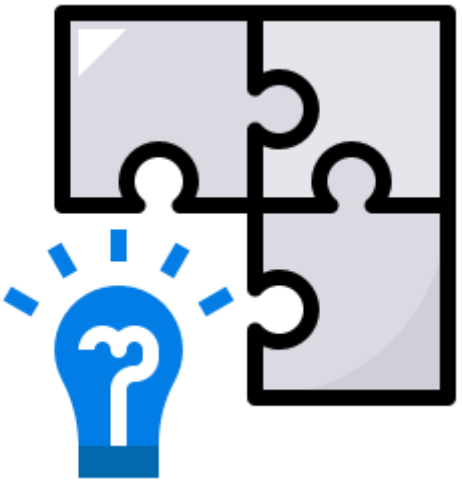
# Algoritmos

O nome algoritmos pode até dar um certo medo, mas não se preocupe, pois é bem fácil de entender. A utilização de um algoritmo facilita na interpretação da lógica, ele é responsável pelo passo a passo de sua aplicação.

Podemos desenvolver um algoritmo de várias maneiras, podendo ser por desenhos, textos, linguagens de desenvolvimento, entre outras opções.

Cada um possui uma lógica, chegar em um resultado significa que você conseguiu atingir objetivo, porém lembre-se que podemos aprimorar nossa lógica aprendendo as outras pessoas. Criar um algoritmo auxilia outras pessoas a compreenderem sua lógica e darem palpites para melhorar a estrutura do sistema.

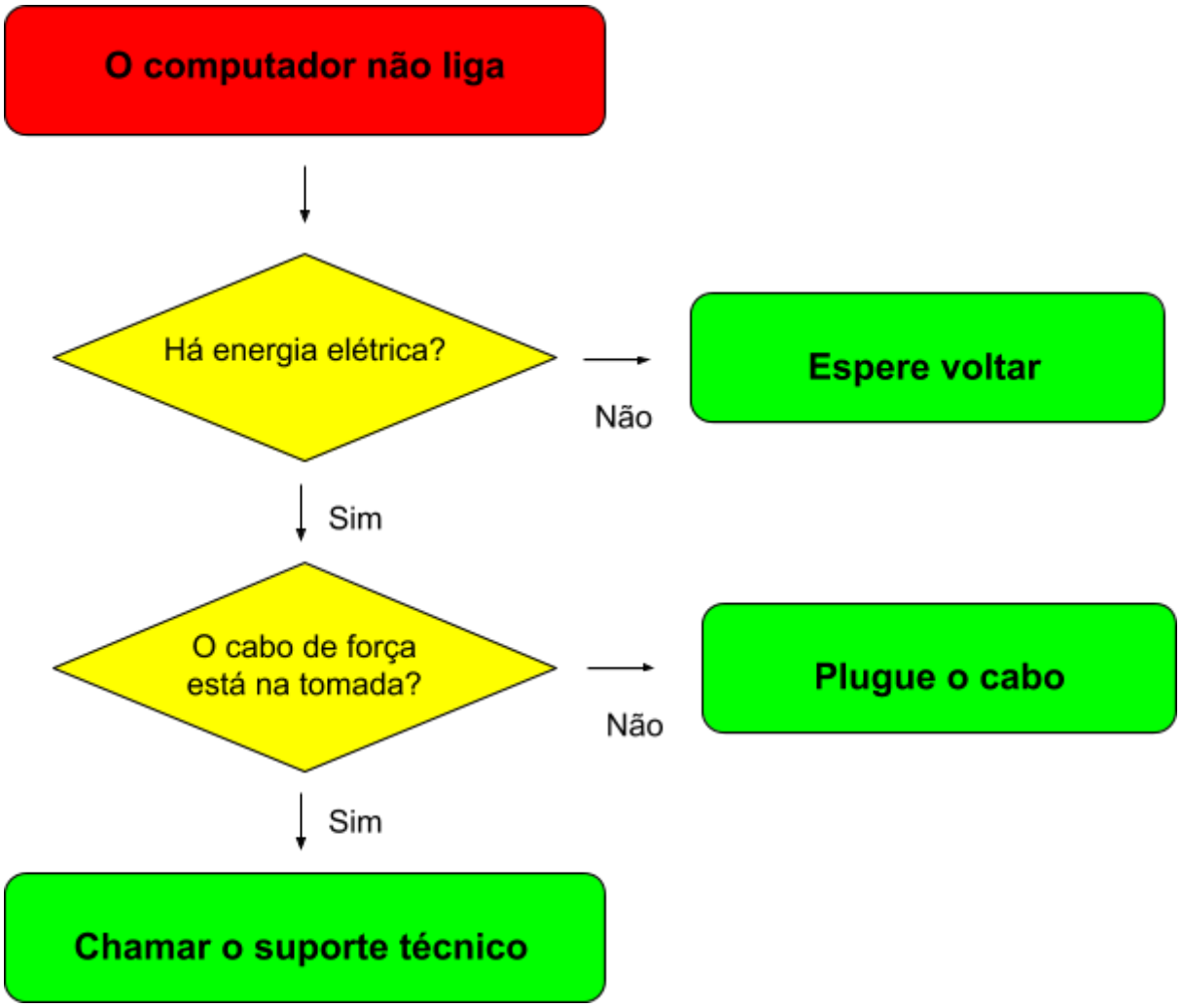
Uma alternativa bem usual é a utilização de fluxogramas, onde formas são utilizadas para ilustrar as fases de um projeto, no próximo capítulo você irá ver como funciona um fluxograma na prática, e poderá utilizar essa técnica nas atividades dispostas em sala ou até mesmo no seu dia a dia como desenvolvedor.



## Fluxograma

Como mencionado no capítulo anterior, o fluxograma é uma maneira muito prática e usual aplicada por muitos desenvolvedores para compreender melhor as funcionalidades de um sistema ou uma parte específica.

Vamos exemplificar o seu uso em um computador que não está ligando:



# Capítulo 02 - Configurando o ambiente

Neste capítulo será abordado sobre nosso ambiente e desenvolvimento, instalando e configurando os softwares necessários.

A configuração do nosso ambiente pode seguir o mesmo padrão para os sistemas operacionais  Windows, Linux e macOS.

## Navegador

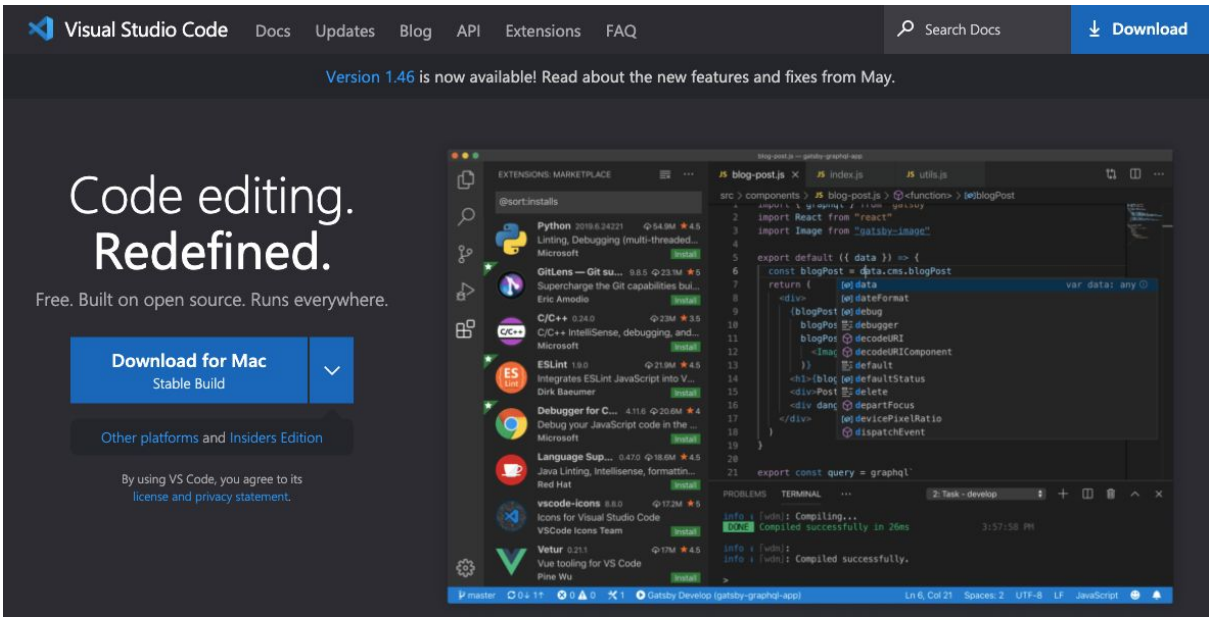
Inicialmente vamos precisar ter um navegador, você pode optar por qualquer um de sua preferência, em nossos exemplos será utilizado o Google Chrome.

O navegador vai servir para testarmos nossas aplicações, pois iremos aprender lógica e algoritmos utilizando linguagens web.

## Instalando o editor

Para desenvolvermos nossos projetos básicos de lógica e algoritmos podemos adotar diversos editores. Tendo como base o mercado atual, será utilizado o Visual Studio Code da Microsoft, uma versão leve, gratuita e disponível nos principais sistemas operacionais.

Este editor possui uma série de complementos interessantes para desenvolvermos em várias linguagens como: Java, C#, PHP, Node.js, Python, JavaScript, TypeScript, entre outras.



# Capítulo 03 - Linguagens utilizadas

Para compreendermos esse conceito de lógica e algoritmos vamos utilizar como base a linguagem de marcação HTML e a linguagem de interação JavaScript, ambas estão integradas aos navegadores, sendo assim não precisa se preocupar em instalar nada para o seu funcionamento.

## Linguagem HTML

O HTML é uma linguagem de marcação, sua função é dispor elementos no navegador. Quando você está em algum site há elementos como: imagens, formulários, textos, tabelas, vídeos, mapas…

Um ponto muito importante é que o HTML não é uma linguagem de programação, mas sim de marcação. Sendo assim não é possível realizar conexões com bancos de dados ou elaborar lógicas utilizando condicionais e laços de repetição por exemplo.

O foco do curso de lógica e algoritmos não é ensinar o HTML, mas sim a lógica na linguagem JavaScript, porém o HTML e o JavaScript precisam estar juntos, sendo assim será passada uma base da linguagem.

# Linguagem JavaScript

O JavaScript é uma linguagem que tem como finalidade criar ações em determinados momentos. Essa linguagem utiliza o padrão de orientação a eventos, sendo assim quando for clicar em algum elemento, digitar, passar o cursor sobre, é possível criar uma rotina que será executada.

Nativamente no navegador o JavaScript consegue fazer muitas coisas que uma linguagem de programação como o Java, C# ou PHP podem fazer, mas há restrições como por exemplo realizar conexões com banco de dados.

Podemos considerar o JavaScript uma linguagem de programação, se estivermos trabalhando com interpretador Node.js, que faz com que o JavaScript consiga ter as mesmas funcionalidades de uma linguagem de programação convencional.

O JavaScript é uma linguagem totalmente gratuita e compreendida diretamente pelo navegador, sem precisar de um outro mecanismo para que suas funcionalidades sejam interpretadas.

Ensinar o JavaScript no módulo de lógica é interessante, pois estruturas como condicionais e laços de repetição por exemplo são idênticas as principais linguagens de programação no mercado atual.

## Exercícios

a) Com base neste capítulo, descreva o que é HTML e o que é JavaScript.

b) É correto afirmar que JavaScript é uma linguagem de programação? Justifique sua resposta.

c) Os conceitos de lógica serão desenvolvidos utilizando a linguagem HTML, pois ela tem suporte a funcionalidades como condicionais e laços de repetição. Essa afirmação está correta? Justifique sua resposta.

# Capítulo 04 - Conceitos básicos

Neste capítulo iremos focar nos conceitos básicos do HTML e do JavaScript na lógica de programação. Os tópicos que serão vistos são:

- Estrutura HTML
- Estrutura JavaScript
- Exibir mensagens
- Variáveis e constantes
- Concatenar dados
- Realizar cálculos
- Operadores relacionais
- Operadores lógicos

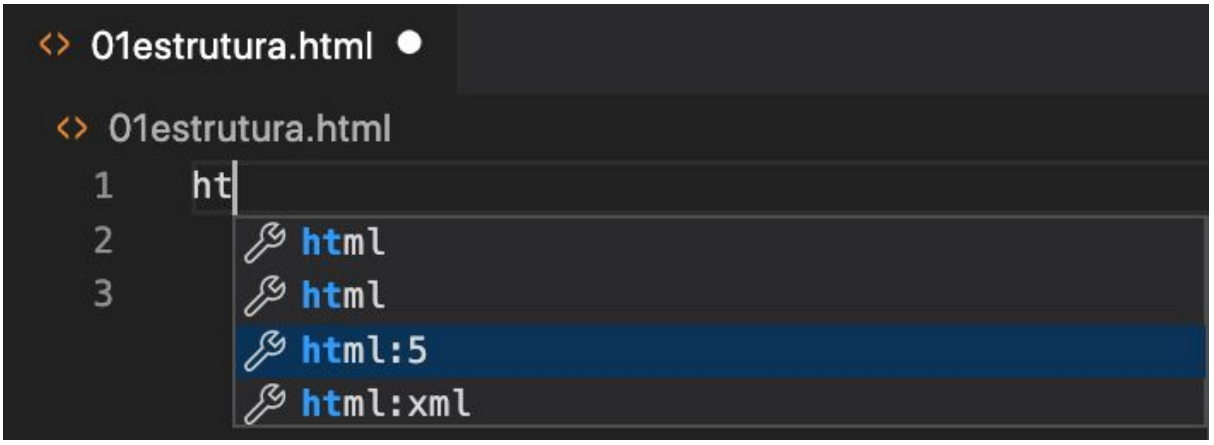
Como pode ver teremos muitos assuntos neste capítulo, sendo assim crie uma pasta com o nome de: **Apex - Lógica**, em seguida abra o Visual Studio Code e selecione a pasta criada (File -> Open Folder) e vamos começar a desenvolver!

## Estrutura HTML

Para compreendermos a estrutura de um arquivo na extensão HTML precisamos criar um arquivo.

No Visual Studio Code procure no menu superior a opção **File**, em seguida clique em **New File**. Será aberto um arquivo de texto simples, agora vá em **File** novamente e selecione **Save**, salve com o nome de **01estrutura.html**.

Quando o arquivo for salvo digite o comando **ht**, espere que a função para autocompletar o código irá aparecer, selecione **html:5** como na imagem abaixo:




Agora você consegue visualizar uma estrutura base HTML, não se assuste, pois serão explicados esses comandos em breve.


Os comandos da linguagem HTML são conhecidos por tags, cada tag é responsável por uma determinada funcionalidade, que pode ser a estrutura de uma tabela, a exibição de uma imagem ou o estilo de um texto.

Abaixo segue uma lista do que faz cada uma das tags HTML criadas anteriormente:

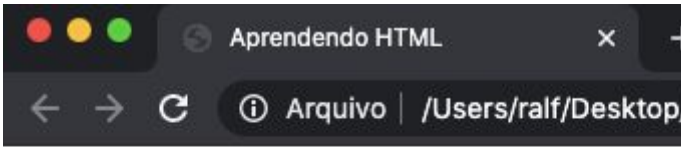
| Tag | Funcionalidade |
|---|---|
| <!DOCTYPE html> | Define que o documento é um HTML |
| <html lang="en"> </html> | Estrutura que estará sua página web |
| <head> </head> | Cabeçalho do site |
| <meta charset="UTF-8"> | Responsável pelos caracteres especiais: ç, é, ã, ê... |
| <meta name="viewport"> | Responsável para trabalhar com conteúdos responsivos |
| <title> </title> | Título do site, que ficará na aba do navegador |
| <body> </body> | Local onde serão exibidos os elementos da página |

Vamos testar nosso código, para isso vamos alterar o título e salvar esse arquivo (File -> Save ou Ctrl + S) .

```
01estrutura.html > ...
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="wid
6       <title>Aprendendo HTML</title>
7   </head>
8   <body>
9
10  </body>
11  </html>
```

Com o arquivo HTML criado e salvo podemos testar, procure o diretório criado para salvar nosso exemplo, haverá um ícone com a imagem de um navegador, clique neste ícone, ou se preferir dê um clique com o botão direito, abrir com e selecione o navegador de sua preferência.

Note que o resultado será uma página totalmente em branco com o nosso título sendo exibido na aba do navegador.

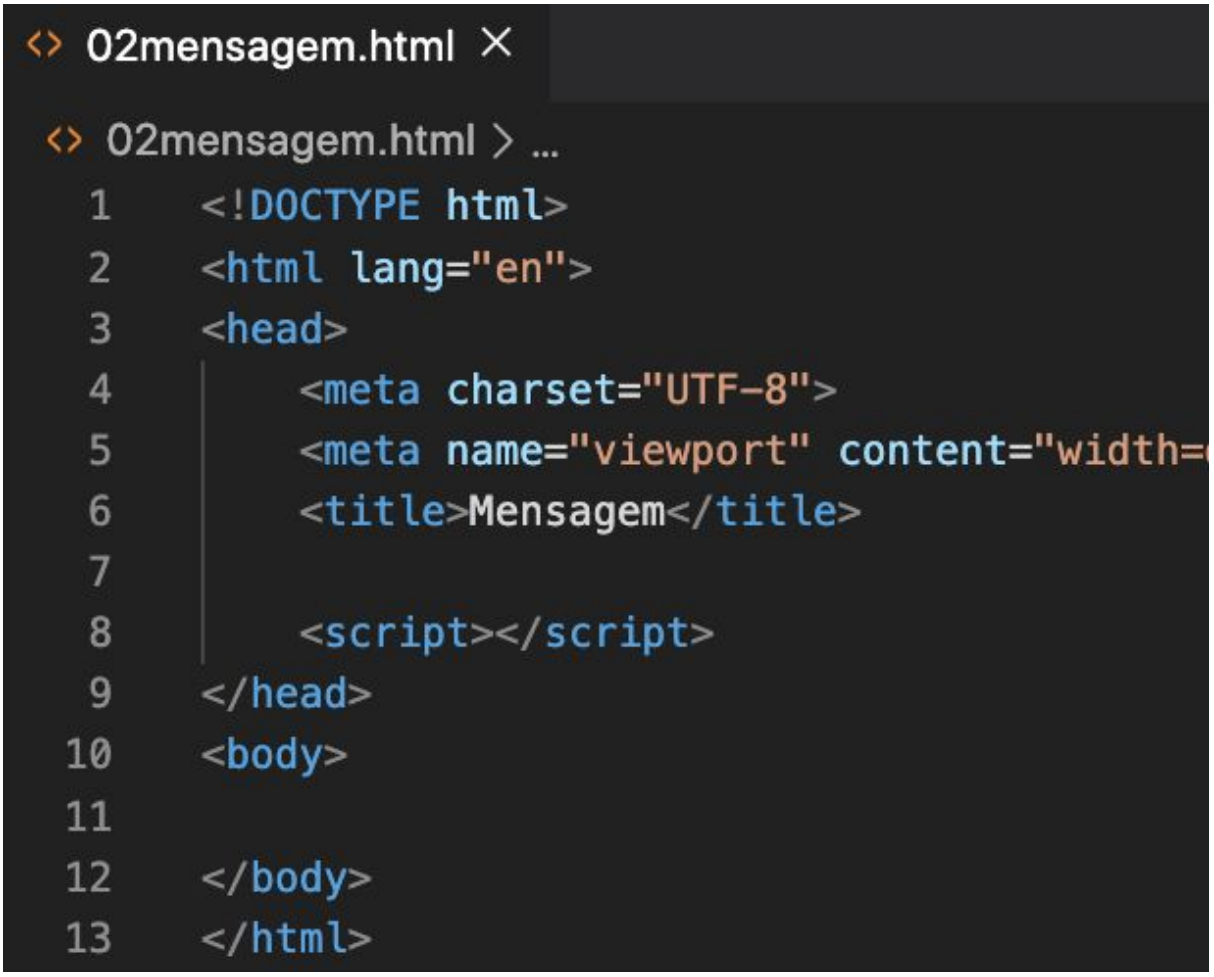



Todos os projetos de lógica irão seguir esse padrão, caso tenha dúvidas converse com o instrutor.

## Estrutura JavaScript

A linguagem JavaScript pode ser compreendida de duas formas, sendo uma delas a criação de um arquivo na extensão **.js** ou internamente em uma página HTML com os comandos **<script></script>**.

Em nossos exemplos vamos utilizar o JavaScript no arquivo HTML, crie um novo arquivo chamado **mensagem.html** com a seguinte estrutura:




Veja que na linha 08 há um novo comando, entre as tags **<script></script>** estarão os códigos Javascript que serão implementados no decorrer dos capítulos.

## Exibindo mensagens

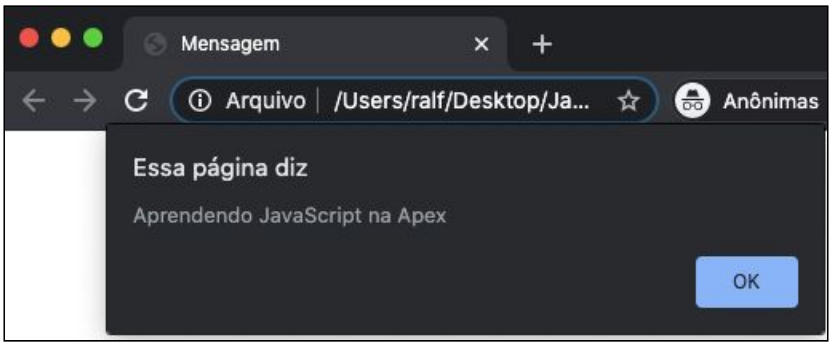
A exibição de textos através de JavaScript é muito tranquila e temos algumas alternativas, siga os exemplos abaixos para aprender a implementar essa funcionalidade:

02mensagem.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device
    <title>Mensagem</title>

    <script>

        // Exibindo texto via alert
        alert("Aprendendo Javascript na Apex")

    </script>
</head>
<body>

</body>
</html>
```

Na linha 10 há um comentário, a utilização de comentários facilita a compreensão de determinadas partes do código. Os comentários não são interpretados pelas linguagens de desenvolvimento, sendo assim o programador não precisa seguir nenhuma regra a não ser a utilização de duas barras como por exemplo: **// Um comentário qualquer…**

Vamos abrir esse arquivo e ver o resultado  no navegador:




O comando **alert()** exibe esse popup, porém não é a única alternativa que podemos utilizar em nossos projetos, abaixo vamos aprender como exibir uma mensagem diretamente no navegador sem ser via o comando **alert()**.

```
8     <script>
9
10        // Exibindo texto via alert
11        //alert("Aprendendo JavaScript na Apex")
12
13        // Exibindo texto via document.write
14        document.write("Lógica e Algoritmos na Apex!")
15
16    </script>
```

A linha 11 foi comentada, sendo assim o comando não será mais executado, agora vamos nos focar nas linhas 13 e 14, a linh 13 possui um comentario, já a linha 14 tem nosso comando **document.write()**, que é responsável por exibir uma mensagem no corpo da página.

## Variáveis e constantes

Variáveis são palavras reservadas pelo desenvolvedor para armazenar dados de maneira temporária, no caso do JavaScript quando a página é atualizada ou o navegador finalizado, as variáveis são removidas da memória.

No JavaScript há duas maneiras de criarmos variáveis, uma utilizando o comando **var** e outra utilizando o **let**. Basicamente as duas maneiras criam variáveis, porém o **let** pertence a uma versão mais recente do JavaScript, já o comando **var** existe desde a criação da linguagem Javascript.

O comando **let** possui uma reutilização de dados, podendo ter uma variável temporária, porém não precisamos nos focar nisso agora, mas é interessante saber que o **let** é muito interessante pensando na reutilização de uma variável.

```
 8    <script>
 9
10        // Criando variáveis com VAR
11        var aluno = "Ralf"
12        var idade = 30
13
14        // Criando variáveis com LET
15        let curso = "Java"
16        let valor = 3000
17
18    </script>
```

Variáveis textuais utilizam aspas, já as numéricas não adicionamos as aspas, essa é uma estrutura padrão utilizada por muitas linguagens. Também a o tipo boolean que chamamos de lógico, nele podemos trabalhar com as informações true e false, muito úteis para realizarmos validações de dados.

## Constantes

As variáveis podem sofrer modificações em diferentes etapas do projeto, já as constantes não podem ser alteradas. Esse tipo de opção é muito interessante quando há valores que o desenvolvedor deseja garantir como inalterável.

```
8       <script>
9
10          // Constante
11          const centro_treinamento = "Apex Ensino"
12
13      </script>
```

Se tentarmos chamar a variável **centro_treinamento** e trocar o nome haverá um erro, pois não é possível alterar o valor de uma constante, o uso de uma constante garante segurança no armazenamento da informação.

## Concatenação

A funcionalidade de uma concatenação é unir informações, podendo ser variáveis, constantes ou palavras fixas. Utilizando o sinal para realizar somas (+), é possível criar essa união entre os dados, veja abaixo o exemplo:

```
8       <script>
9
10          //  Variáveis
11          var nome = "Mayra"
12          var idade =  31
13
14          // Mensagem concatenando os dados
15          document.write("Olá "+nome+" você tem "+idade+" anos.")
16
17      </script>
```

# Cálculos

Um desenvolvedor de sistemas trabalha constantemente com operações matemáticas, sendo assim é necessários fazermos uma breve explicação sobre o que podemos fazer em uma linguagem de programação. Observe a tabela abaixo os cálculos disponíveis:

| Símbolo | Cálculo |
|---|---|
| + | Somar |
| - | Subtrair |
| * | Multiplicar |
| / | Dividir |
| % | Operação modular ou mod (resto da divisão) |

Vamos compreender melhor na prática, veja abaixo o uso dos cálculos acima e seus respectivos resultados:

```
8   <script>
9
10      // Soma
11      document.write(5 + 5)  // Retorna 10
12
13      // Subtração
14      document.write(9 - 3)  // Retorna 6
15
16      // Multiplicação
17      document.write(5 * 4)  // Retorna 20
18
19      // Divisão
20      document.write(10 / 2) // Retorna 5
21
22      // Operação Modular
23      document.write(6 % 2)  // Retorna 0
24
25  </script>
```

## Operadores relacionais

Provavelmente você já os viu nas aulas de matemática, os operadores relacionais servem para realizarmos comparativos entre valores, podendo ser informações do tipo textuais, numéricas ou lógicos. A tabela abaixo irá especificar quais operadores podemos utilizar:

| Operador | Significado |
|---|---|
| > | Maior |
| >= | Maior ou igual |
| < | Menor |
| <= | Menor ou igual |
| == | Igual |
| != | Diferente |

## Operadores lógicos

Os operadores lógicos possuem uma característica interessante para verificarmos dois ou mais valores ou negar alguma informação por exemplo, veja abaixo a tabela com o operador e a sua descrição:

| Operador | Finalidade |
|---|---|
| && | Verifica se dois ou mais valores são verdadeiros |
| \|\| | Verifica se ao menos uma verificação é verdadeira |
| ! | Negação |

## Exercícios

Antes de prosseguirmos vamos testar seus conhecimentos, com base nesse capítulo resposta às questões abaixo:

a) O que é uma variável?

b) Qual a finalidade de criarmos uma constante?

c) O que é uma operação modular?

d) Explique e exemplifique o uso de uma concatenação.

e) Qual a finalidade de utilizar um operador relacional?

# Capítulo 05 - Estrutura de condição

Neste capítulo iremos abordar o uso de condicionais, em especial esse capítulo deve ser compreendido muito bem, pois será a base para desenvolvermos qualquer tipo de sistema.

Utilizamos a expressão **SE** para verificarmos qual ação tomar, além disso podemos implementar com outra expressão que é a **SENÃO**, que irá executar quando uma condicional **SE** não for verdadeira.

## Condicional simples

A estrutura de uma condicional simples realiza uma ação quando uma verificação for verdadeira. Vamos supor que iremos pedir uma idade e será exibida uma mensagem se a idade for de 18 ou mais:

```
8     <script>
9
10        // Variável contendo uma idade
11        var idade = 20
12
13        // Condicional
14        if(idade >= 18){
15            document.write("Maior de idade")
16        }
17
18    </script>
```

Note que temos uma variável com o valor de 20 na linha 11, a condicional é composta por parênteses **()** que validam uma ação a ser feita que esteja entre as chaves **{}**.

## Condicional composta

As condicionais compostas têm a finalidade de fornecer uma resposta caso a validação seja positiva ou uma resposta se falsa. Condicionais simples possuem apenas uma resposta, já condicionais duas.

Vamos implementar o exemplo utilizado para compreender a estrutura de uma condicional simples, implementando uma resposta caso o indivíduo seja menor de idade:

```
8   <script>
9
10      // Variável contendo uma idade
11      var idade = 11
12
13      // Condicional
14      if(idade >= 18){
15          document.write("Maior de idade")
16      }else{
17          document.write("Menor de idade")
18      }
19
20  </script>
```

Utilizando o comando **else** criamos uma ação caso a condicional **if** seja falsa, note que diferente do **if** não realizamos nenhuma verificação utilizando os parênteses, pois o uso do **else** é automático quando a condicional **if** é falsa.

## Condicional aninhada

Até aqui conseguimos compreender que uma condicional executa uma ação dependendo se a verificação for verdadeira ou falsa, mas e se necessitarmos criar mais que duas ações? Neste caso utilizamos as condicionais aninhadas, vamos exemplificar esse conceito da seguinte maneira:

```
8   <script>
9
10      // Variável contendo uma linguagem de programação
11      var linguagem = "Java"
12
13      // Condicional
14      if(linguagem == "Java"){
15          document.write("Linguagem multiplataforma da Oracle")
16      }else if(linguagem == "C#"){
17          document.write("Linguagem desenvolvida pela Microsoft")
18      }else if(linguagem == "Python"){
19          document.write("Linguagem ideal para ciência de dados")
20      }else if(linguagem == "PHP"){
21          document.write("Linguagem para desenvolvimento web")
22      }else{
23          document.write("Ops! Tente novamente")
24      }
25
26  </script>
```

Na linha 11 foi criada uma variável chamada **linguagem** contendo o termo **Java**. Já na linha 14 começa nossa verificação, nesse exemplo temos uma frase para as linguagens: Java, C#, Python e PHP, caso seja informado algo diferente dessas quatro linguagens o comando **else** executa a linha 23.

Note que iniciamos com o **if**, adicionamos vários **else if** e apenas um **else**, com isso podemos entender que sempre devemos começar com um **if**, já os **else if** podem ter quantos quisermos e podemos ter um **else** se desejarmos.

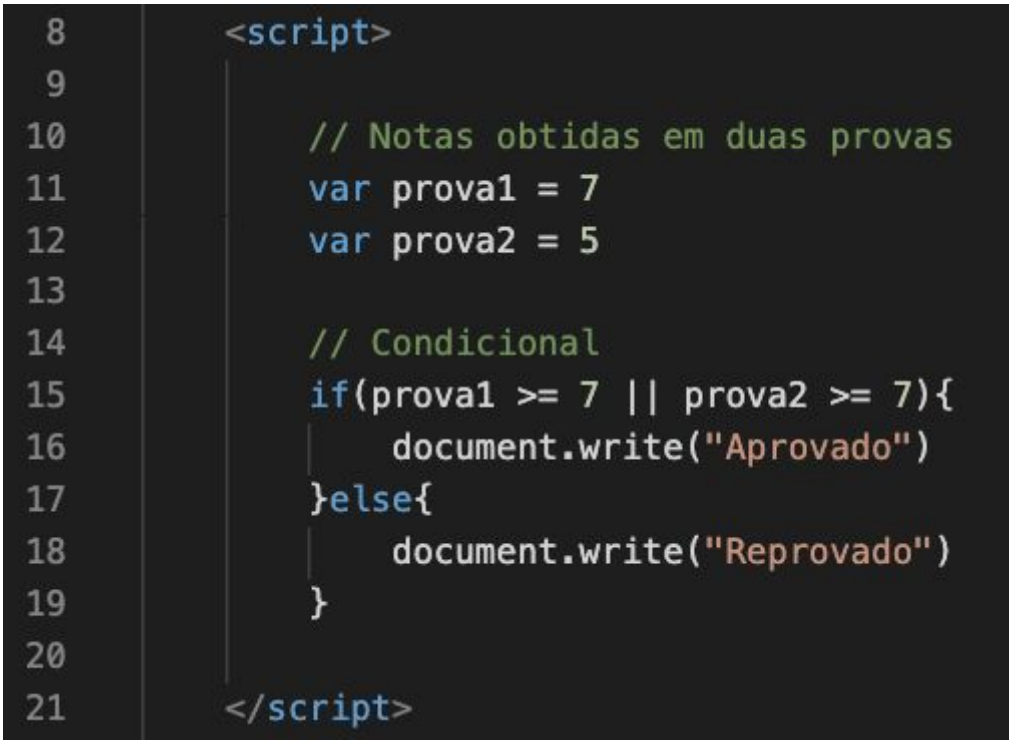
## Condicionais e operadores lógicos

Já estudamos sobre operadores lógicos, porém não implementamos nenhum exemplo, então agora você vai aprender a utilizar operadores lógicos com condicionais. Essa prática é muito normal no dia a dia dos desenvolvedores, no nosso exemplo abaixo vamos supor que teremos uma média e um número máximo de faltas que um aluno pode ter. Se a média for maior ou igual a 7 **e** tiver até 10 faltas estará aprovado, caso contrário reprovado.

```
8   <script>
9
10      // Variável contendo a média
11      var media = 8.5
12
13      // Variável contendo as faltas
14      var faltas = 5
15
16      // Condicional
17      if(media >= 7 && faltas <= 10){
18          document.write("Aprovado")
19      }else{
20          document.write("Reprovado")
21      }
22
23  </script>
```

O operador lógico **e** (&&) garante que a ação da condicional **if** será executada somente se toda as verificações forem verdadeiras, caso uma seja falsa automaticamente será executada a próxima instrução de verificação (else if) ou a ação que estiver no **else** (se houver).

Também há o operador **ou** que podemos utilizar para validar duas ou mais informações, esse operador é constituído por dois pipes (||), vamos exemplificar o seu uso na imagem abaixo:




Nas linhas 11 e 12 há duas variáveis para representar as notas obtidas em dois trabalhos avaliativos, já na linha 15 há nossa condicional. Note o uso do operador **ou**, se o conseguir nota 7 ou mais em pelo menos uma prova, estará aprovado, caso contrário reprovado.

O operador lógico **ou** é mais maleável que o operador lógico **e**, com o tempo você vai aprender melhor quando utilizar cada um deles.

## Exercícios

a) Peça duas notas, em seguida realize a média e informe se o aluno está aprovado ou reprovado. Para essa atividade, o aluno precisa ter média 7 ou mais para ser considerado aprovado.

b) O usuário informa dois valores numéricos, se forem iguais deverão ser somados, caso contrário multiplicados.

c) Haverá uma lista com quatro produtos que uma lanchonete vende, o usuário precisa informar o código do produto, em seguida o valor pago por ele. Nosso sistema precisa retornar o troco que deve ser dado ao cliente, valide caso o cliente informe um pagamento inferior ao valor do pedido.

| Código | Pedido | Valor |
|---|---|---|
| 1 | Pizza | R$ 12,00 |
| 2 | Pão de Queijo | R$ 4,00 |
| 3 | Macarrão | R$ 16,00 |

d) Peça para o usuário informar três números, em seguida exiba o menor deles.

e) O usuário deverá informar a largura de cada lado de um triângulo, peça as três larguras (direito, inferior e esquerdo) e verifique o tipo de triângulo. Os tipos e triângulos são:

| Tipo | Especificação |
|---|---|
| Isósceles | Dois lados iguais e um diferente |
| Escaleno | Três lados diferentes |
| Equilátero | Três lados iguais |

# Capítulo 06- Estrutura de escolha

A estrutura de escolha ou **switch - case**, é uma maneira mais elegante para avaliarmos ações que podem ocorrer em determinadas partes de um sistema.

Essa técnica é muito parecida com o uso de condicionais, porém é mais enxuta e não podemos utilizar operadores lógicos ou relacionais.

## Compreendendo o funcionamento

Para compreender a elaboração de um **switch - case** ou estrutura de escolha, vamos exemplificar utilizando como base uma variável cidade contendo o valor Blumenau, em seguida teremos algumas opções de retorno.

```
8   <script>
9
10      // Variável contendo uma cidade
11      var cidade = "Blumenau"
12
13      // Estrutura de escolha
14      switch(cidade){
15          case "Blumenau":
16          document.write("Cidade da Oktoberfest")
17          break
18
19          case "Joinville":
20          document.write("Maior cidade de Santa Catarina")
21          break
22
23          case "Florianópolis":
24          document.write("Capital de Santa Catarina")
25          break
26
27          default:
28          document.write("Cidade não encontrada")
29      }
30
31  </script>
```

## Exercícios

a) O usuário informa uma temperatura, em seguida poderá selecionar o tipo de conversão (Celsius e Fahrenheit).

b) Crie um conversor de moedas. Será informado o valor e as conversões:
   - Real para Dólar
   - Dólar para Real
   - Real para Euro
   - Euro para Real

c) O usuário informa o nome de um produto, valor e segmento (vestuário, eletrônico ou alimentício). Se o segmento for vestuário será dado 5% de desconto sobre o valor do produto, caso eletrônico 8% e alimentício 10%.

d) Crie um sistema de votação para três candidatos. O eleitor poderá selecionar um dos três nomes, em seguida será contabilizado o voto para o candidato.

e) O usuário informa o horário atual no Brasil (apenas hora), em seguida informa um país. Retorne o horário daquele país se baseando pela hora informada. Cadastre ao menos três países diferentes.

# Capítulo 07 - Laços de repetição

Neste capítulo iremos aprender a trabalhar com estruturas de repetição, daremos ênfase em três tipos de laços que são: **while**, **do - while** e **for**.

Podemos trabalhar com várias funcionalidades utilizando os laços de repetição, como por exemplo: listar informações de bancos de dados, verificar uma cadeia de caracteres como nomes, e-mails e telefones, além de filtrarmos informações através de vetores, arquivos de textos e demais tipos de dados.

## Incrementadores e decrementadores

Os incrementadores e decrementadores são úteis para trabalharmos com índices que os laços se baseiam para saber em qual posição estão.

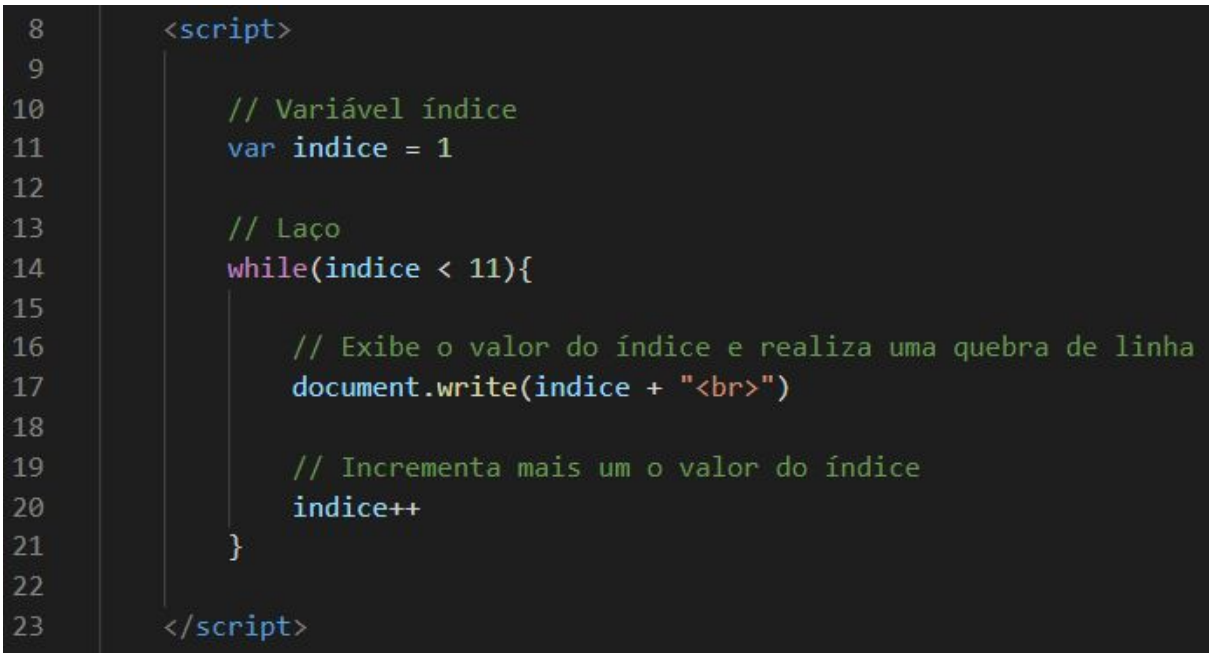
Veremos que um laço terá um ponto inicial e um ponto final, muitas vezes podemos utilizar os incrementadores ou decrementadores como base para realizarmos ações pré definidas.

Abaixo há uma lista exibindo como podemos trabalhar com esses incrementos e decrementos:

| Símbolo | Ação |
|---|---|
| ++ | Incrementa mais um |
| -- | Decrementa menos um |
| +=2 | Incrementa somando por dois |
| -=3 | Decrementa subtraindo por três |
| *=5 | Incrementa multiplicando por cinco |
| /=2 | Decrementa dividindo por dois |

## Laço enquanto

Laço enquanto ou **while**, esse laço é o mais simples e usual, veja o exemplo abaixo criando um contador de 1 a 10:



```
8   <script>
9
10      // Variável índice
11      var indice = 1
12
13      // Laço
14      while(indice < 11){
15
16          // Exibe o valor do índice e realiza uma quebra de linha
17          document.write(indice + "<br>")
18
19          // Incrementa mais um o valor do índice
20          indice++
21      }
22
23  </script>
```

Na imagem acima podemos compreender a estrutura de um laço **while**, na linha 11 foi criada uma variável contendo o número 1, que terá como finalidade o ponto inicial do nosso laço. Repare que na linha 14 há uma verificação de que enquanto o índice for menor que 11 serão executados os comandos entre as chaves. Na linha 17 exibimos o valor do índice e concatenamos com o comando **<br>**, que é uma quebra de linha na linguagem HTML, por fim na linha 20 incrementamos o índice em mais um número. Quando chegar na linha 21 que é a finalização do laço, nosso **while** fará novamente a verificação do índice, agora valendo 2 e executar novamente todo o processo, até que o valor do índice chegue a 11, parando o processo de exibição da mensagem e incremento do índice.

## Laço faça enquanto

O laço faça enquanto ou **do while**, é muito parecido com a estrutura de repetição **while**. O principal diferencial é a ordem de execução, enquanto o laço **while** realiza uma validação para executar uma ação, o **do while** obrigatoriamente realiza uma ação e depois verifica se será executada novamente aquela ação.

Para compreender sua estrutura vamos exemplificar pedindo vários nomes até que o termo **finalizar** seja informado.

```
8     <script>
9
10        // Variável nome
11        var nome
12
13        // Laço
14        do{
15
16            // Pede um nome e armazena na variável
17            nome = prompt("Informe um nome")
18
19        }while(nome != "finalizar")
20
21    </script>
```

Na linha 11 foi criada uma variável **nome**, essa variável poderá ser manipulada nas linhas posteriores podendo armazenar nomes e validar o laço de repetição. Na linha 14 iniciamos o laço com o comando **do**, sendo assim a ação é executada imediatamente, na linha 17 a variável **nome** é utilizada para armazenar um nome informado pelo usuário utilizando a função **prompt**, que é uma caixa de mensagem com a possibilidade de digitar algo. Para finalizar nossa estrutura de laço, na linha 19 realizamos uma verificação com a variável **nome**, enquanto o nome informado for diferente do termo finalizar, o laço será executado infinitamente.

## Laço para

Mais um comando de laço, agora vamos aprender a utilizar o laço para ou **for**. Esse laço é muito utilizado para manipular vetores, em breve veremos o uso de vetores no desenvolvimento de sistemas.

O laço de repetição **for** possui uma estrutura de verificação diferenciada dos laços **while** e **do while**, veja o exemplo abaixo exibindo os valores de 10 a 1:

```
8   <script>
9
10      // Laço de repetição
11      for(var indice = 10; indice >= 1; indice--){
12
13          // Exibir o valor do índice e pular linha
14          document.write(indice + "<br>")
15
16      }
17
18  </script>
```

Na linha 11 temos nosso laço **for** constituído por três parâmetros, foi criado um índice tendo o valor inicial 10, em seguida o segundo parâmetro é a finalização do laço, neste caso o laço irá finalizar quando o índice for menor que 1, e por fim temos o terceiro parâmetro que é o decremento do índice.

Nos laços **while** e **do while** o incremento e decremento ficam dentro das chaves, porém no laço **for** essa característica será informada entre os parênteses.

Na linha 14 temos a ação de mensagem e uma quebra de linha, fazendo com que seja exibido o índice de 10 até 1. É importante destacar que utilizamos um decremento para esse exemplo, mas você pode utilizar um incremento se for necessário.

## Exercícios

a) O usuário irá informar um número, em seguida realizar a tabuada daquele número.

b) O usuário informa um número inicial e um número final, em seguida retorne a quantidade de pares e ímpares daquela cadeia de valores.
Vamos supor que os números informados foram: 6 e 12, sendo assim serão 4 pares (6, 8, 10 e 12) e 3 ímpares (7, 9 e 11).

c) Crie um pequeno sistema de votação, exiba em um **prompt()** as opções:

| |
|---|
| 1 - Alessandra |
| 2 - Caio |
| 3 - Geraldo |
| 4 - Talita |
| 5 - Sair |

Enquanto não for digitado o número 5 o laço deverá realizar a ação de **prompt()** para pedir o voto e contabilizar para o candidato. Quando informado o número 5 retorne a quantidade de votos que cada candidato obteve.

d) Crie uma estrutura de laços de repetição que exiba a seguinte estrutura:
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1

# Capítulo 08 - Vetores

Vetor ou **array** é uma variável que possibilita armazenar um grande número de dados, podemos comparar o uso de vetores com bancos de dados, só que de maneira temporária. Neste capítulo iremos aprender a como manipular dados em diferentes estruturas de vetores.

## Estrutura do vetor

Um vetor ou **array** trabalha com o padrão de uma linha e várias colunas, para exemplificar imagine que queremos criar uma variável contendo cinco nomes, visualmente sua estrutura ficaria assim:

| | | | | |
|---|---|---|---|---|
| Ana | Breno | Cleber | Daniela | Elaine |

Temos um vetor com cinco posições, para manipularmos um vetor precisamos compreender que cada nome ficará em um índice, esse índice por padrão é iniciado com zero, sendo assim nosso vetor com o índice ficará assim:

| | | | | |
|---|---|---|---|---|
| Ana | Breno | Cleber | Daniela | Elaine |
| **0** | **1** | **2** | **3** | **4** |

A grande vantagem no uso do vetor é que podemos criar uma única variável e armazenar vários dados, sem o uso do vetor precisaríamos criar uma variável para cada informação, no exemplo acima precisamos criar cinco variáveis, uma para cada nome, imagina se fosse em um sistema com mil nomes? Seria algo bem complexo para se trabalhar, mas com o uso do vetor esse trabalho se torna mais fácil.

## Exemplo prático

Nesta parte iremos exemplificar o uso de um vetor utilizando o JavaScript, vamos criar um vetor de nomes e exibir cada um deles:

```
 9      <script>
10
11          // Criar um vetor com nomes
12          var nomes = ["Larissa", "Marcelo", "Natália", "Orlando", "Paloma"]
13
14          // Laço
15          for(var indice=0; indice<nomes.length; indice++){
16              document.write(indice+" "+nomes[indice]+"<br>")
17          }
18
19      </script>
```

Vamos compreender a estrutura do código na imagem acima, na linha 12 há um vetor chamado nomes, que entre os colchetes há cinco nomes. Na linha 15 utilizamos o laço de repetição **for** que possui três parâmetros, sendo eles uma variável índice iniciando do zero, pois todo o vetor começa do zero, em seguida um **nomes.length**, que retorna a quantidade de dados no vetor de **nomes**, neste caso são cinco nomes e por fim o incremento que será mais um para cada volta. Na linha 16 temos a exibição do índice, além da exibição do nome na posição do índice e uma quebra de linha, o resultado no navegador será esse:

# Funções para vetores

Agora que você conseguiu compreender a estrutura de um **array**, podemos trabalhar com algumas funcionalidades úteis como: contagem, cadastro, alteração e exclusão de dados que compõem nossos vetores.

Veja a imagem abaixo algumas ações que podem ser feitas:

```
9   <script>
10
11      // Criar um vetor com nomes
12      var nomes = ["Larissa", "Marcelo", "Natália", "Orlando", "Paloma"]
13
14      // Adicionar um novo nome
15      nomes.push("Rebeca")
16
17      // Alterar Paloma para Patrícia
18      nomes[4] = "Patrícia"
19
20      // Removendo a Natália
21      nomes.splice(2, 1)
22
23      // Exibir nomes
24      for(var indice=0; indice<nomes.length; indice++){
25          document.write(indice+" "+nomes[indice]+"<br>")
26      }
27
28      // Informar a quantidade de nomes cadastrados
29      document.write("Há "+nomes.length+" nomes no vetor")
30
31  </script>
```

Vamos compreender desenvolvido na imagem acima:

- Linha 12: Criamos um vetor com cinco nomes.
- Linha 15: Adicionamos o nome Rebeca no vetor utilizando a função **push**.
- Linha 18: Utilizando a posição entre colchetes, podemos atribuir um novo nome.
- Linha 21: O método **splice** remove um dado do vetor, para isso é necessário passar dois parâmetros que são: a posição do elemento que desejamos remover e quantos elementos a partir daquela posição serão removidos.
- Linha 24: Laço de repetição para exibir os cinco nomes dispostos no vetor.
- Linha 29: Exibição da quantidade de elementos no vetor utilizando a função **length**.

# Exercícios

a) Crie um vetor contendo cinco cidades, em seguida exiba seus valores.

b) Desenvolva um sistema que o usuário informa cinco números através do comando **prompt**, em seguida exiba os valores informados e mostre a soma desses valores.

c) Crie um laço que peça nomes, poderão ser armazenados vários nomes, não há uma quantidade mínima nem máxima. Quando informada a palavra **sair**, exiba todos os nomes e informe a quantidade de nomes cadastrados.

d) O usuário informa cinco números, em seguida exiba na ordem contrário que foram informados.

e) Faça com que sejam pedidos e armazenados cinco números. Informe quantos números dez foram informados pelo usuário.

f) Peça sete números, em seguida exiba a soma desses valores, a média e quantos números são maiores ou iguais a média.

g) Haverá um cardápio contendo dez itens (você define o nome e o valor de cada item), cada item terá um código de 1 a 10, além do número 11 que finaliza o sistema. Enquanto não for digitado o número 11 será contabilizado em uma lista, quando digitado o número 11 exiba os produtos adquiridos e o total dessa compra.

h) Crie um vetor que peça cinco números, em seguida exiba os cinco números informados e mostre qual é o menor número e o maior número.

i) Desenvolva um sistema para gerenciar cursos, as funcionalidades pedidas são:
   1) **Cadastrar curso** - Pede o nome do novo curso
   2) **Selecionar cursos** - Exibe todos os cursos
   3) **Remover curso** - Pede o nome do curso e realiza a exclusão
   4) **Sair do sistema** - Finaliza o laço

# Capítulo 09 - Matriz

Vetor multidimensional, matriz ou simplesmente **array** é denominado um conjunto maior de dados. Anteriormente quando vimos o uso de um vetor unidimensional havia uma linha com várias colunas, agora podemos ter várias linhas e várias colunas.

Achou confuso? Fica tranquilo, o conceito não é difícil e praticando ficará ainda mais fácil compreender o uso de uma matriz.

Vamos imaginar que precisamos criar uma agenda para trabalhar com quatro contatos, nela podemos ter os dados: nome, idade e cidade, sua estrutura ficará assim:

| | | |
|---|---|---|
| Felipe | 22 | Blumenau |
| Gabriela | 17 | Brusque |
| Henrique | 13 | Jaraguá do Sul |
| Isabela | 26 | Indaial |

Na tabela acima temos uma matriz que chamamos de 4x3, onde o primeiro número corresponde a linha e o segundo número a coluna. Para manipular uma informação precisamos trabalhar com linhas e colunas, vamos compreender da seguinte maneira:

a) O que tem na primeira linha, segunda coluna?
**R: O valor é 22.**

b) O que tem na posição 1,2?
**R: Brusque.**

c) O que tem na posição 0,0?
**R: Felipe, pois a primeira linha e primeira coluna sempre começam do zero.**

d) Qual é a localização da cidade de Indaial?
**R: Localizada na linha 3, coluna 2.**

## Exemplo prático

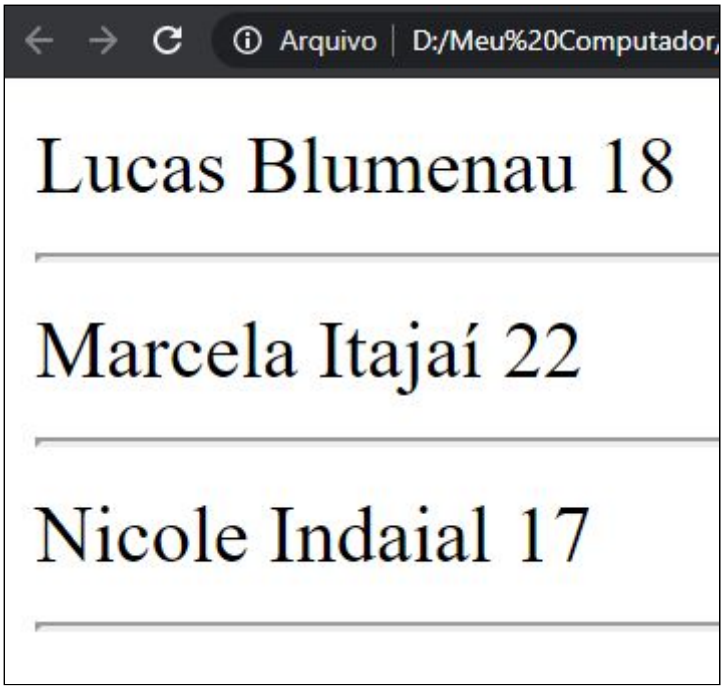
Para compreender melhor o uso de uma matriz, será criada uma pequena lista de contatos, contendo nome, cidade e idade e exibindo esses dados:

```
 9    <script>
10
11        // Criar uma matriz
12        var dados = [
13            ["Lucas", "Blumenau", 18],
14            ["Marcela", "Itajaí", 22],
15            ["Nicole", "Indaial", 17]
16        ]
17
18        // Laço de linha
19        for(var linha=0; linha<dados.length; linha++){
20
21            // Laço de coluna
22            for(var coluna=0; coluna<3; coluna++){
23
24                // Exibir o valor da matriz
25                document.write(dados[linha][coluna]+" ")
26
27            }
28
29            // Cria uma nova linha para cada loop realizado
30            document.write("<hr>")
31        }
32
33    </script>
```

Na linha 12 criamos nossa matriz, cada linha ficará entre colchetes, e cada coluna será separado por vírgula. Na linha 19 será criado um laço para percorrer cada linha do vetor, será utilizado como base um laço visto no capítulo anterior. Na linha 22 há um outro laço, pois em cada linha há três colunas, e esse laço será responsável por exibir o valor de cada coluna.

Veja que na linha 25 realizamos a exibição e um espaçamento entre os dados, para exibir o dado correspondendo utilizamos a estrutura: **matriz[][]**, o primeiro colchetes manipula a posição da linha e o segundo a coluna. Para finalizar na linha 30 há um comando para gerar uma linha em nosso arquivo html, assim conseguiremos visualizar melhor a lista de contatos.

Na figura abaixo você conseguirá visualizar o resultado de nosso laço de repetição manipulando nossa matriz de contatos:




Com essa base podemos desenvolver sistemas mais complexos, **arrays** são considerados assuntos difíceis, então vale a pena treinar bastante. Utilize o tempo em sala para desenvolver as atividades e se possível desenvolva as atividades propostas neste capítulo.

# Exercícios

a) Crie uma matriz 3x3, onde a primeira coluna terá o nome de um produto, a segunda coluna a marca e a terceira o valor. Esses dados não precisam ser informados pelo usuário, essas informações podem ser pré-definidas como no exemplo de contatos, assim que desenvolver a matriz, exiba os dados utilizando laços de repetição.

b) Implemente uma matriz 3x2, na primeira coluna serão armazenados nomes de cursos, já na segunda coluna os valores. Ao término liste os dados informados para armazenar na matriz.

c) Crie uma matriz 4x4 e peça para o usuário informar 16 números. Quando informados os 16 números exiba cada um deles, além da soma e a média dos valores.

d) Crie um sistema para uma agenda de contatos, esse sistema terá um menu com as seguintes funcionalidades:
   1) Cadastrar contato
   2) Selecionar contatos
   3) Alterar contato
   4) Remover contato
   5) Sair do sistema

e) Elabore um caixa de mercado. Haverá uma lista de produtos que você irá criar, cada produto terá um nome e um valor. Inicialmente o caixa poderá selecionar um código que representa um produto e a quantidade comprada pelo cliente, enquanto não digitar o número zero, essa ação será repetida inúmeras vezes. Quando digitado o número zero exiba cada produto comprado, além de sua quantidade, valor unitário, valor total daquele produto (quantidade * valor unitário) e por fim o total da compra.

# Capítulo 10 - Funções

A utilização de funções facilita o desenvolvedor a separar as funcionalidades de um sistema, além de poder reutilizar ações, evitando os códigos repetitivos. Abaixo segue um exemplo simples de função para exibirmos uma mensagem:

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width,
6       <title>Funções</title>
7
8       <script>
9
10          // Função de mensagem
11          function mensagem(){
12              alert("Utilizando funções")
13          }
14
15      </script>
16
17  </head>
18  <body>
19
20      <!-- Botão para executar a função -->
21      <button onclick="mensagem()">Clique aqui</button>
22
23  </body>
24  </html>
```

Na linha 11 criamos uma função através do comando **function**, toda a função necessita de um nome e uma estrutura de parênteses **()** para informarmos parâmetros, utilizamos em outro exemplo os parâmetros. Na linha 21 há um botão com um evento de clique, esse evento chama nossa função de mensagem que exibe um popup com uma frase.

# Eventos

Para trabalharmos com funções podemos chamar através de eventos nos elementos HTML, sendo assim abaixo veremos alguns dos eventos que podemos utilizar:

| Ação | Funcionamento |
|---|---|
| onClick | Executa uma função quando clicar em um elemento HTML |
| onChange | Executa uma função quando um valor é alterado |
| onKeyUp | Executa uma função quando digitar |
| onLoad | Executa uma função quando o elemento é carregado |
| onSubmit | Executa uma função quando um formulário envia dados |

Na linguagem JavaScript há outros tipos de eventos que podemos implementar, porém nosso foco é ensinar a lógica, então nas atividades propostas você poderá utilizar esses cinco eventos.

## Exemplo prático

Vamos supor que precisamos criar um sistema para trabalhar com a média de um aluno, esse sistema irá pedir duas notas, realizar a média, verificar a situação (aprovado ou reprovado) e exibir o resultado.

Vamos por partes, criando um arquivo HTML com um pequeno formulário:

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width,
6       <title>Funções</title>
7   </head>
8   <body>
9
10      <!-- Formulário -->
11      <form>
12          <input type="text" placeholder="Primeira nota">
13          <input type="text" placeholder="Segunda nota">
14          <input type="button" value="Verificar">
15      </form>
16
17  </body>
18  </html>
```

Na imagem anterior temos um formulário bem simples, contendo uma tag **form**, duas caixas de texto pedindo as notas e um botão. Vamos adicionar um identificador (**id**) nos dois campos de texto e utilizar o evento de clique no botão.

```
10      <!-- Formulário -->
11      <form>
12          <input type="text" placeholder="Primeira nota" id="nota1">
13          <input type="text" placeholder="Segunda nota" id="nota2">
14          <input type="button" value="Verificar" onclick="acao()">
15      </form>
```

Nosso formulário está pronto, agora podemos implementar as funções, veja na imagem abaixo a estrutura para fazermos esse exemplo funcionar:

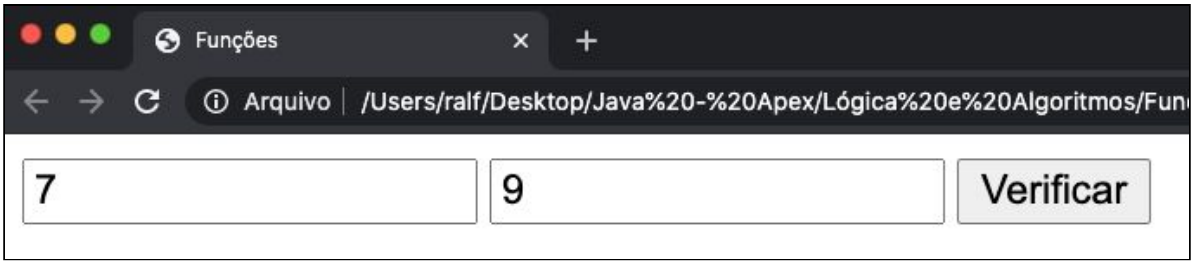
```
8       <script>
9
10          // Evento quando clicado no botão
11          function acao(){
12
13              // Obter as duas notas
14              var nota1 = parseInt(document.getElementById("nota1").value)
15              var nota2 = parseInt(document.getElementById("nota2").value)
16
17              // Obter a média
18              var media = calculo(nota1, nota2)
19
20              // Obter a situação
21              var situacao = verificarSituacao(media)
22
23              // Mensagem
24              alert("Aluno está "+situacao+" com a média "+media)
25
26          }
27
28          // Retornar a média
29          function calculo(nota1, nota2){
30              return (nota1 + nota2) / 2
31          }
32
33          // Retornar a situação
34          function verificarSituacao(media){
35              return media >= 7 ? "aprovado" : "reprovado"
36          }
37
38      </script>
```

Na linha 11 criamos nossa função responsável pelo clique no botão que está no formulário, nas linhas 14 e 15 vamos obter as notas através do identificador, implementamos os **ids** para sua localização nos campos de texto que estão no formulário.

Para realizarmos a média criamos uma nova função que está na linha 29, ela pede dois parâmetros que são **nota1** e **nota2**, o retorno será a soma dividido por dois. A ideia dessa função é separar o cálculo de outras funções.

Quando executada a função **calculo()**, a variável **media** que está na linha 18 terá o resultado obtido pela função. Agora com a média podemos obter a situação que é verificada através de uma função na linha 34 chamada **verificarSituacao()**, pedindo o parâmetro média irá realizar um operador ternário que retorna se está aprovado ou reprovado.

Agora na linha 21 teremos a situação do aluno, agora podemos exibir através de um alerta, para isso na linha 24 há uma concatenação. Observe o resultado nas imagens abaixo:





Note que ao invés de desenvolvermos um método para obter dados, realizar o cálculo e validar a situação, criamos três métodos diferentes, esse tipo de prática é muito normal na programação, pois facilita na reutilização de funções e também na manutenção do código.

## Exercícios

a) Crie um formulário, onde o usuário irá informar dois valores e uma operação de soma, subtração, multiplicação ou divisão.

Haverá as seguintes funções:

1) Ação - Responsável por obter os dados informados, selecionar um cálculo e exibir
2) Soma - Haverá dois parâmetros que retornará a soma dos valores
3) Subtração - Haverá dois parâmetros que retornará a subtração dos valores
4) Multiplicação - Haverá dois parâmetros que retornará a multiplicação dos valores
5) Divisão - Haverá dois parâmetros que retornará a divisão dos valores

b) Criar uma agenda para manipular os dados: nome, e-mail, idade e cidade, essa agenda terá opções para cadastrar, alterar, selecionar e excluir contatos. Cada ação da agenda deverá estar em uma função.

c) Implemente um sistema para cadastrar produtos contendo os seguintes dados: nome do produto, marca, valor, quantidade em estoque. As funcionalidades que esse sistema deverá ter são:

- Cadastrar produtos
- Selecionar produtos
- Alterar produtos
- Excluir produtos
- Pesquisar produtos através do nome

Algumas regras para esse sistema:

- Não pode haver produtos com o mesmo nome
- Não poderão ser cadastrados com valor zero ou negativos
- Quando cadastrado um produto, a quantidade em estoque precisa ser de pelo menos um.