

# Algoritmos Naturais

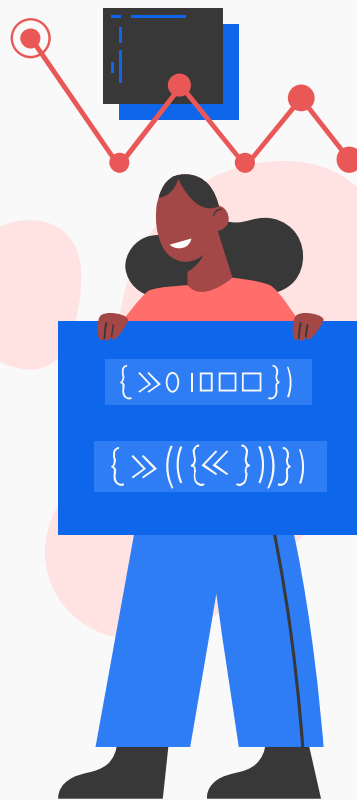
## Desenvolver Algoritmos

Msc. Lucas G. F. Alves  
e-mail: [lgfalves@senacrs.com.br](mailto:lgfalves@senacrs.com.br)



# Planejamento de Aula

Revisão JavaScript e Variáveis  
Algoritmos naturais e estruturados  
Comandos de entrada e saída  
Representação Visual  
Atividade Avaliativa A



# JavaScript

Uma linguagem de programação é um conjunto de normas (sintaxe) que permite criar comandos para o computador

Baixo

Alto

Nível de Abstração

Binário

Assembly

C,  
C++

Java, Python,  
**JavaScript**

{((({>>}))<<}

{ }



# JavaScript

[ ]

**JavaScript** é uma linguagem moderna e aceita por todos os navegadores mais usados hoje em dia.

Usaremos, agora no início do curso, o nosso navegador (indicamos Chrome ou Firefox) para ver os códigos que criamos

Um navegador precisa de uma página HTML para rodar o seu código, então veremos agora de maneira muito simples como criar uma página

{ }

{((({>>}))<<}

-[ ]



# JavaScript

[ ]

## Começando um Projeto

Precisamos ter a extensão **live server** instalada no **VSCode** e criar uma pasta com dois arquivos:

**index.html** ⇒ Primeiro arquivo que o navegador olha.

**index.js** ⇒ Arquivo onde escreveremos nosso código JS.

Obs: os dois arquivos devem estar na mesma pasta!

{ }

{((({>>}))<<}

-[ ]



# JavaScript



## Começando um Projeto

Começamos colocando o código padrão do HTML (se você apertar os botões ! + **enter** o VSCode faz esse código pra você!)

Adicionamos uma linha de código, dentro da tag head, que vai ligar nosso arquivo **index.js** ao HTML

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



```
{((({>>}))<<}
```



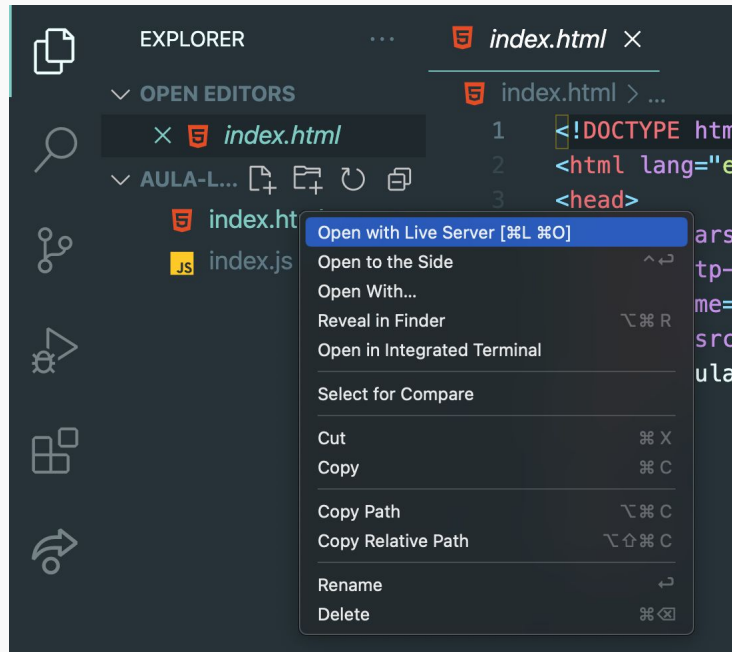
# JavaScript

## Começando um Projeto

Para abrir o html com o Live Server:

Clique com o botão direito em cima do arquivo index.html;

Depois clique em Open with Live Server ou Abrir com Live Server;





# Sintaxes Basicas

[ ]

## Comentários

São estruturas que permitem escrevermos textos que serão ignorados para executar o programa.

Eles devem começar com `//` **ISSO É UM COMENTÁRIO** ou estarem entre

`/*`

Um comentário é ignorado no momento em que o programa é rodado

`*/`

{ }

{((({>>}))<<}

-[ ]





# Sintaxes Basicas



[ ]

## Imprimindo no console

O JS possui uma sintaxe específica para imprimir informações no console do navegador.

```
console.log("Olá Mundo!")
```

{ }

```
{ ((({ >> } )) << }
```

- [ ]



# Sintaxes Basicas

[ ]

## Imprimindo no console

O JS possui uma sintaxe específica para imprimir informações no console do navegador.

```
console.log("Olá Mundo!")
```

## Pedindo informações para o usuário

Em aplicações Web, conseguimos pedir que o usuário nos passe alguma informação, assim:

{ }

```
prompt("Qual é o seu nome?")
```

```
{((({>>}))<<}
```

- [ ]



# Sintaxes Basicas

[ ]

## Resumo

Para trabalhar com JS, vamos usar um arquivo index.html e um index.js.

Os dois arquivos devem estar na mesma pasta.

Para linkar o arquivo JS ao HTML, usamos a tag `<script src="/index.js" defer></script>`.

Comentários: de linha `//` e de bloco `/* */`.

{ }

Imprimir uma info: `console.log()`.

Solicitar uma info do usuário: `prompt()`.

`({{{({>>}})}<<}`

- [ ]



# Variáveis

[ ]

O que são?

Variáveis são estruturas que permitem guardar e acessar quaisquer informações no nosso código



Funcionam como **gavetas** (guardar e acessar)

Gaveta B recebe o **arquivo A**,  
mas o **Arquivo A** não é a gaveta B

{ }

{((({>>}))<<}

-[ ]



# Variáveis

[ ]

## Sintaxes

Antes de usarmos estas variáveis, nós precisamos declará-las (criá-las).

```
const novaVariavel = 100
//   ^           ^   ^   ^
// declaração nome atribuição valor
```

Devemos escolher nomes significativos.

Nomes não podem começar com números ou caracteres especiais.

{ }

Utilizamos o padrão **camelCase**.

- primeira letra minúscula;
- primeira letra entre uma palavra e outra é maiúscula;

```
{((({>>}))<<}
```

- [ ]



# Variáveis

[ ]

## Sintaxes

**const:** quando uma variável é declarada usando const, nós dizemos que ela é constante.

O seu valor **NÃO** pode mudar ao longo do programa.

```
const idade = 22  
idade = 24
```

Errado

**let:** quando uma variável é declarada usando let, ela **PODE** ter seu valor alterado.

{ }

```
let idade = 22  
idade = 24
```

( { ( ( { >> } ) ) << }

- [ ]



# Variáveis

[ ]

## Sintaxes

Podemos imprimir mais de uma coisa no console separando elas por vírgula.

Será adicionado um espaço entre as palavras.

```
const nome = "Fulana"
```

```
const idade = 21
```

```
console.log("Olá!", "Meu nome é ", nome, "e eu tenho", idade, "anos")
```

```
{ } // Olá! Meu nome é Fulana e eu tenho 21 anos
```

```
{ ((({ >> } )) << ) }
```

- [ ]



# Tipos de Variáveis

[ ]

Os valores que as variáveis do JS assumem possuem tipos. Hoje apresentaremos três deles:

1  
2  
4.50  
-300  
56089

Numbers

{ }

“Marina”  
“Abobrinha”  
“31”

Strings

true  
false

Booleans

{((({>>}))<<}

- [ ]





# Tipos de Variáveis

[ ]

**Numbers:** são os tipos que representam números

```
const idade = 23
```

```
const altura = 1.79
```

```
const temperatura = -20
```

**Strings:** são os tipos que representam conjunto de caracteres (texto)

```
const nome = "Pedro"
```

```
let idade = "23"
```

{ }

( { ( ( { >> } ) ) << }

- [ ]



# Tipos de Variáveis

[ ]

## Booleans

George Boole foi o inventor do que chamamos de álgebra booleana.

Esta álgebra leva em consideração que os valores assumidos são somente:

- TRUE ou 1;
- FALSE ou 0;

```
{ } let souUmBoolean = true  
souUmBoolean = false
```



{ ((({ >> } )) << ) }

- [ ]



# Exercícios

[ ]

Faça os seguintes itens:

- 1) Crie uma variável e atribua seu primeiro nome.
- 2) Crie uma variável e atribua seu sobrenome.
- 3) Crie uma variável e atribua sua idade.
- 4) Crie uma variável que diga se você é ou não estudante.
- 5) Imprima o seu nome, sobrenome, idade e status de estudante no console.

{ }

```
{((({>>}))<<}
```

- [ ]



# Resumo

[ ]

Usamos **let** para declarar variáveis que podem ter seu valor alterado no decorrer do programa.

**const** para variáveis que terão valores constantes.

**Strings** representam textos.

**Numbers** representam números.

**Booleanos** são tipos que recebem apenas dois valores: verdadeiro (true) ou falso (false).

{ }

( { ( ( { >> } ) ) << }

- [ ]



# Tipos de Variáveis

[ ]

Como descobrir o tipo da variável.

**typeof**: comando que permite ver o tipo do valor da variável.

```
const got = "Game Of Thrones"  
const temporadasDeGot = 8
```

```
typeof got //string  
typeof temporadasDeGot //number
```

{ }

{((({>>}))<<}

-[ ]



# Tipos de Variáveis

[ ]

**undefined**

Tipo que representa a falta de valor de uma variável.

```
let novaVariavel  
typeof novaVariavel //undefined
```

```
novaVariavel = 2  
typeof novaVariavel //number
```

```
{ } novaVariavel = undefined  
typeof novaVariavel //undefined
```

{((({>>}))<<}

-[ ]



# Tipos de Variáveis

[ ]

null

**null**: também representa a falta de valor da variável.

Existem algumas diferenças entre undefined e null, e uma delas é que o null precisa ser atribuído diretamente a uma variável.

```
let minhaVariavel  
console.log(minhaVariavel) //undefined  
minhaVariavel = null  
{ } console.log(minhaVariavel) //null
```

{((({>>}))<<}

-[ ]



# Exercícios

[ ]

- 1) Peça o nome do usuário através do prompt e guarde em uma variável.
- 2) Peça a idade do usuário através do prompt e guarde em uma variável.
- 3) Veja qual é o tipo das variáveis de nome e idade.

{ }

{((({>>}))<<}

-[ ]





# Conversão entre Tipos

[ ]

Como vimos no exercício anterior, tudo o que o usuário insere em um prompt é uma string!

Podemos fazer a conversão entre esses dois tipos usando métodos fornecidos pelo Javascript!

Número ⇒ String:

toString()

```
const idadeNumero = 23
```

```
const idadeTexto = idadeNumero.toString()
```

```
console.log(typeof idadeNumero)
```

```
console.log(typeof idadeTexto)
```

{ }

String ⇒ Número:

Number()

```
const idadeTexto = "23"
```

```
const idadeNumero = Number(idadeTexto)
```

```
console.log(typeof idadeTexto)
```

```
console.log(typeof idadeNumero)
```

```
{ ((({ >> } )) << ) }
```

{ }



# Resumo

[ ]

Conseguimos criar comentários usando `//` ou `/**/`

`console.log(mensagem)` gera uma mensagem no console. `prompt()` solicita ao usuário que insira uma informação.

Variáveis declaradas com **const** não mudam enquanto as criadas com **let** podem mudar.

**Numbers:** representam números.

**Strings:** representam texto.

**Boolean:** são tipos que recebem apenas dois valores: verdadeiro (true) ou falso (false).

**typeof:** permite ver o tipo do valor de uma variável.

Conversões entre tipos:

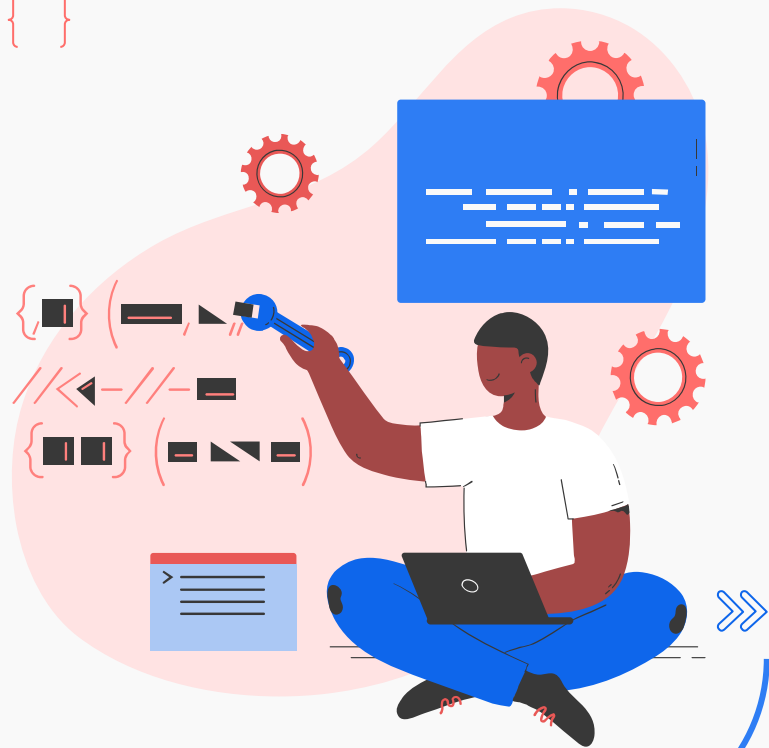
{ }

Número  $\Rightarrow$  String: `toString()`

String  $\Rightarrow$  Número: `Number()`

{((({>>}))<<}

-{ ] }





# Algoritmos naturais x estruturados



## Diferenças

A principal diferença reside na forma como as instruções são organizadas e executadas:

**Algoritmos Naturais:** São mais fáceis de compreender, como uma receita culinária tradicional. As instruções são escritas como fala, com passos sequenciais e pode ter alterações.



**Algoritmos Estruturados:** São um pouco mais difíceis de entender, como um código de programação. As instruções são organizadas em blocos lógicos, com sequência definida e condições específicas para cada passo.

{((({>>}))<<}





# Algoritmos naturais x estruturados



[ ]

## Algoritmos Naturais

Algoritmo Natural (receita de bolo):

2 ovos,  
1 xícara de leite,  
2 xícaras de farinha de trigo,  
1 xícara de açúcar.

Misture os ingredientes secos.  
Bata as claras em neve.  
Adicione as claras à massa.  
Asse em forno pré-aquecido.  
Espere esfriar antes de servir.

{ }

{((({>>}))<<}

-[ ]



# Algoritmos naturais x estruturados



## Algoritmo Estruturado

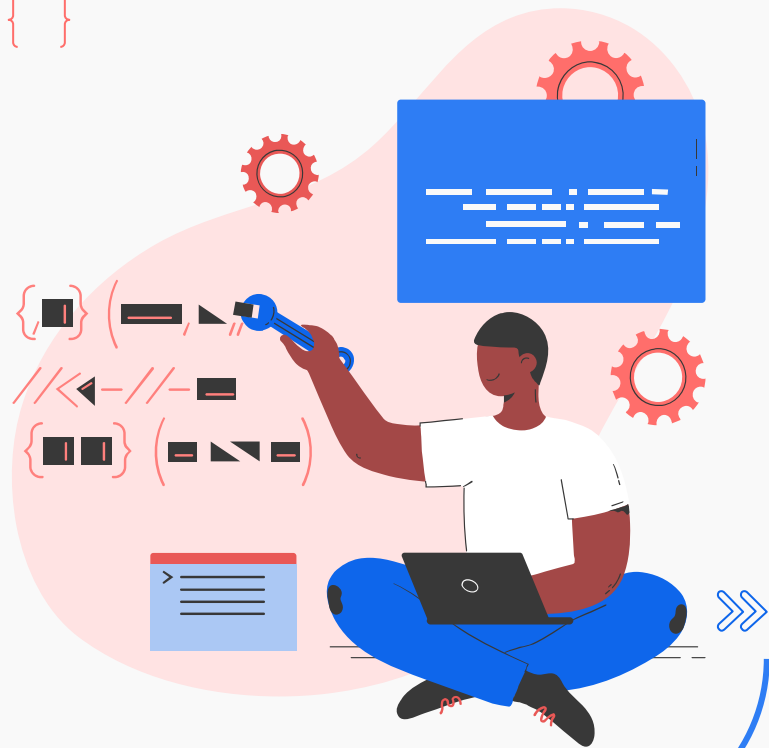
Algoritmo Estruturado (código para calcular a média):

1. Leia as notas dos alunos.
2. Calcule a soma das notas.
3. Divida a soma pelo número de alunos.
4. Exiba a média.



```
{((({>>}))<<}
```







# Comandos de entrada

[ ]

## O que são Inputs ou Entrada?

Os comandos de entrada em uma receita, são os ingredientes que você entrega ao cozinheiro.

No mundo da computação, isso significa fornecer informações ao computador, como:

- Digitar texto no teclado.
- Clicar com o mouse.
- Ler dados de um arquivo.
- Receber informações de outros dispositivos.

{ }

{((({>>}))<<}

-[ ]





# Comandos de entrada

[ ]

## O que são Inputs ou Entrada?

Os comandos de entrada em uma receita, são os ingredientes que você entrega ao cozinheiro.

No mundo da computação, isso significa fornecer informações ao computador, como:

- Digitar texto no teclado.
- Clicar com o mouse.
- Ler dados de um arquivo.
- Receber informações de outros dispositivos.

{ }

Exemplo de código: `prompt("Qual é o seu nome?")`

```
{((({>>}))<<}
```

-[ ]

# Comandos de processamento





# Comandos de processamento

[ ]

O que são processamento?

Os comandos de processamento em uma receita, é o trabalho do cozinheiro transformando os ingredientes na comida.

No computador, é o processamento das informações que você forneceu, como:

- Realizar cálculos matemáticos.
- Comparar dados e tomar decisões.
- Ordenar informações.
- Criar novos dados a partir dos existentes.

{ }

Exemplo de código: let somar = 1+1;

```
{((( {>> } ))<< ) }
```

- [ ]

# Comandos de saída





# Comandos de saída

[ ]

## O que são saídas ou Outputs?

Os comandos de saída em uma receita, é a comida pronta que o cozinheiro te entrega.  
No computador, as saída são as informações resultantes do processamento, que podem ser:

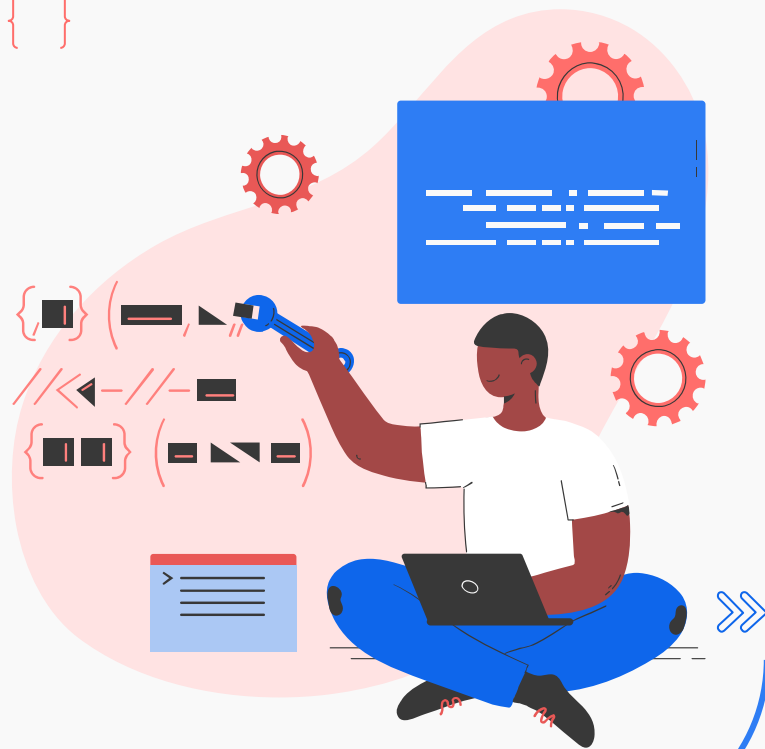
- Exibidas na tela.
- Salvas em um arquivo.
- Enviadas para outro dispositivo.
- Impressas em papel.

{ }

Exemplo de código: `console.log(somar);`

`({{{({>>}}))<<}`

- [ ]





# Representação visual



## Formas de representar algoritmos

Existem diversas formas de representar um algoritmo. As mais comuns são por textos livres, fluxogramas e pseudo-códigos.

### Texto livre

Texto livre seria o algoritmo natural descrito em linguagem natural. Exemplo de uma receita de bolo.



{((({>>}))<<}





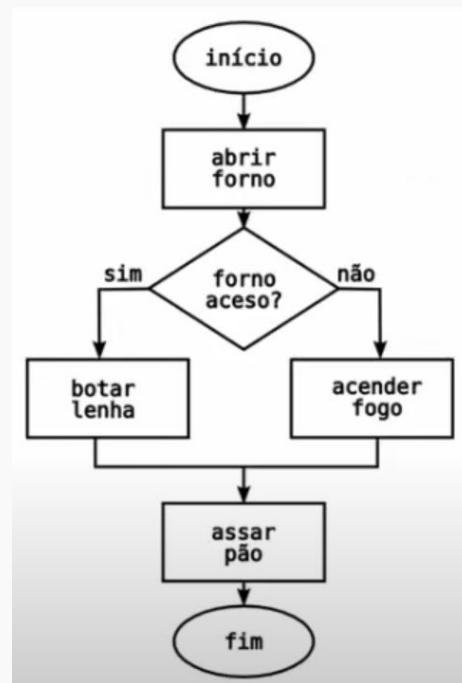
# Representação visual



## Fluxograma

É a representação gráfica de algoritmos onde formas geométricas diferentes Implicam em ações (instruções ou comandos) diferentes.

Um fluxograma é um diagrama que nos ajuda a entender a ordem em que cada coisa acontece na nossa solução de uma forma visual.



{((({>>}))<<}


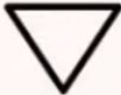




-[]





# Representação visual

## Fluxograma

SIMBOLOGIA DE FLUXOGRAMAS PADRÃO ANSI - American National Standards Institute			
SÍMBOLO	SIGNIFICADO	SÍMBOLO	SIGNIFICADO
	Operação		Armazenagem
	Movimento / Transporte		Sentido de fluxo
	Ponto de decisão		Conexão¹
	Inspeção		Limites (início, pare, fim)
	Documento impresso	1 -	Utilizado quando o fluxograma não cabe em uma única página
	Espera		

{((({>>}))<<}







# Atividade em grupo

[ ]

## Desvendando o Mistério da Biblioteca - Algorithm Flowchart

Criar um fluxograma na ferramenta miro para um algoritmo que encontre um livro raro na biblioteca. Você sabe que o livro está na biblioteca, mas não sabe em qual estante ele está.

Pistas:

- O livro é sobre história.
- O livro tem mais de 100 anos.
- O livro tem uma capa vermelha.

Exemplo:

{ } Retângulo: "Ir para a seção de história da biblioteca."

Losango: "O livro está na estante?"

Seta: Conecta as etapas do fluxograma.

{((( { >> } )) << ) }

- [ ]



# Dúvidas?

[ ]



{ }

{{{{}}

- [ ]

# Obrigado!



E-mail: [lgfalves@senacrs.com.br](mailto:lgfalves@senacrs.com.br)



{({({ >> } ) ) << }

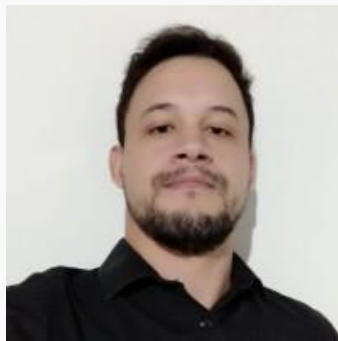


(( { >> 0 i □ □ □ } ))

```
((: 00 - =>> } )  
{ (<1 00 1 000 >> )}  
((: 0)>"< )  
<01 001} +100 0}>  
((: 0)>"< )  
{ (<1 00 1 000 >> )}
```



# Professor



**Lucas G. F. Alves**

