



HELLO  
WORLD



# VALIDAÇÃO DE FORMULÁRIOS COM REGEN

# ESTRUTURA BÁSICA DE UM FORMULÁRIO HTML

Antes de começar a manipulação com JavaScript, precisamos entender a estrutura básica de um formulário.

```
1 <form id="form-cadastro">
2   <label for="nome">Nome:</label>
3   <input type="text" id="nome" name="nome" required />
4
5   <label for="email">E-mail:</label>
6   <input type="email" id="email" name="email" required />
7
8   <label for="senha">Senha:</label>
9   <input type="password" id="senha" name="senha" required />
10
11  <button type="submit">Cadastrar</button>
12 </form>
13 <div id="mensagem"></div>
```

# CAPTURANDO O EVENTO DE ENVIO DO FORMULÁRIO

Precisamos capturar o evento de envio do formulário para prevenir o envio padrão e validar os dados.



```
1 const formCadastro = document.getElementById("form-cadastro");  
2  
3 formCadastro.addEventListener("submit", function (event) {  
4     event.preventDefault(); // Previne o envio do formulário  
5     validarFormulario();  
6 });
```



# EXPRESSIONES REGULARES



## O que são Expressões Regulares (Regex)?

Expressões Regulares, ou Regex (do inglês Regular Expressions), são padrões de texto usados para buscar, validar, ou manipular strings de forma eficiente. Elas são amplamente utilizadas em linguagens de programação, editores de texto, bancos de dados e muitas outras ferramentas.

### Por que usar Expressões Regulares?

- Validação de dados: Verificar se um e-mail, número de telefone ou CPF está no formato correto.
- Busca e substituição: Encontrar palavras em um texto e substituí-las por outras.
- Extração de informações: Extrair partes específicas de um texto, como datas ou URLs.

#### 1. Caracteres Literais

São letras, números ou símbolos que representam exatamente o que aparecem.

Exemplo:

- `a` encontra a letra "a" em um texto.
- `123` encontra o número "123" em um texto.



## 2. Metacaracteres

São símbolos especiais que têm um significado específico em Regex.

Principais metacaracteres:

Metacaractere	Significado	Exemplo
.	Qualquer caractere, exceto nova linha	<code>a.b</code> encontra <code>a*b</code> , <code>a3b</code>
^	Início da string	<code>^a</code> encontra "a" apenas no início
\$	Fim da string	<code>b\$</code> encontra "b" no final
*	Zero ou mais ocorrências	<code>a*</code> encontra <code>aaaa</code> , <code>a</code> ou vazio
+	Uma ou mais ocorrências	<code>a+</code> encontra <code>a</code> , <code>aa</code>
?	Zero ou uma ocorrência	<code>a?</code> encontra <code>a</code> ou vazio
\	Escapa um metacaractere	<code>\.</code> encontra <code>.</code> literal
[]	Conjunto de caracteres	<code>[abc]</code> encontra <code>a</code> , <code>b</code> ou <code>c</code>
		Ou lógico
()	Agrupamento	<code>(ab)+</code> encontra <code>ab</code> , <code>abab</code>

### 3. Exemplos Práticos de Regex

#### Validação de CPF



```
1 ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$
```


#### Explicação:

- ^: Início da string.
- [a-zA-Z0-9.\_%+-]+: Caracteres permitidos antes do @.
- @: Símbolo obrigatório.
- [a-zA-Z0-9.-]+: Nome do domínio.
- \.: Ponto literal.
- [a-zA-Z]{2,}: Extensão do domínio com no mínimo 2 letras.
- \$: Fim da string.



A pixel art illustration of a city at night. On the left is a tall, slender skyscraper with a blue base and a yellow and red grid pattern. To its right is a shorter, wider building with a blue base and a yellow and red grid pattern. In the center background is a suspension bridge with a blue frame and yellow and red grid pattern. On the right is a tall, rectangular building with a blue base and a yellow and red grid pattern. The sky is dark blue with several small, white, pixelated clouds. The text "USANDO REGEH" and "NO JAVASCRIPT" is displayed in a large, white, pixelated font with a red outline, centered in the upper half of the image.

USANDO REGEH  
NO JAVASCRIPT



Em JavaScript, Regex pode ser usado com os métodos `test()`, `match()`, `replace()`, entre outros.

1. Validar uma string com `test()`

```
1 const email = "exemplo@email.com";
2 const regex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
3
4 if (regex.test(email)) {
5   console.log("E-mail válido!");
6 } else {
7   console.log("E-mail inválido!");
8 }
```



## 2. Substituir texto com replace()



```
1 const texto = "Olá Mundo";  
2 const resultado = texto.replace(/\s/g, "-"); // Substitui espaços por hífens  
3 console.log(resultado); // Olá-Mundo
```

## 3. Extrair dados com match()



```
1 const texto = "Meu telefone é 123-456-7890";  
2 const regex = /\d{3}-\d{3}-\d{4}/;  
3 const telefone = texto.match(regex);  
4 console.log(telefone); // 123-456-7890
```

# PRINCIPAIS VANTAGENS DO USO DE REGEK

01

**Eficiência:** Executa buscas e substituições complexas rapidamente.

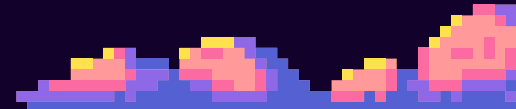
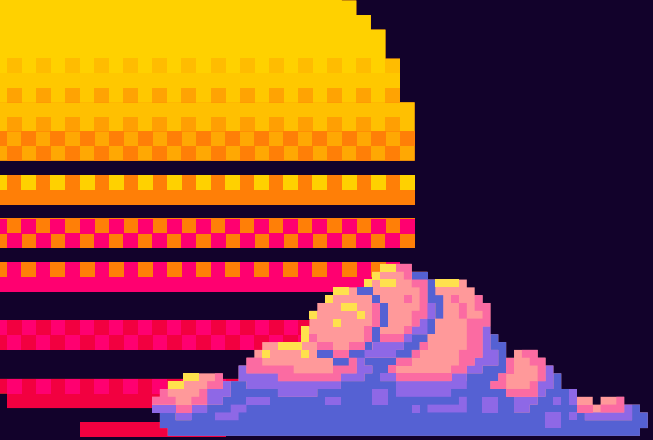
02

**Flexibilidade:** Pode ser adaptado a diferentes cenários e formatos de dados.

03

**Compatibilidade:** Funciona em várias linguagens de programação.





# DICAS PARA USAR REGEX

01

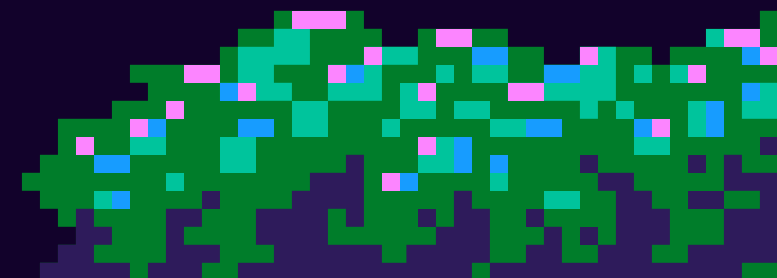
**Teste suas expressões:** Utilize ferramentas como [Regex101](#) para testar e depurar.

02

**Evite Regex muito complexas:** Prefira soluções simples e legíveis.

03

**Comente sua Regex:** Em projetos maiores, explique o que cada parte faz.



# EXERCÍCIO PRÁTICO

**Tarefa:** Crie um formulário com os seguintes campos e implemente a validação.

01

Nome de usuário (apenas letras e números, mínimo 5 caracteres).

02

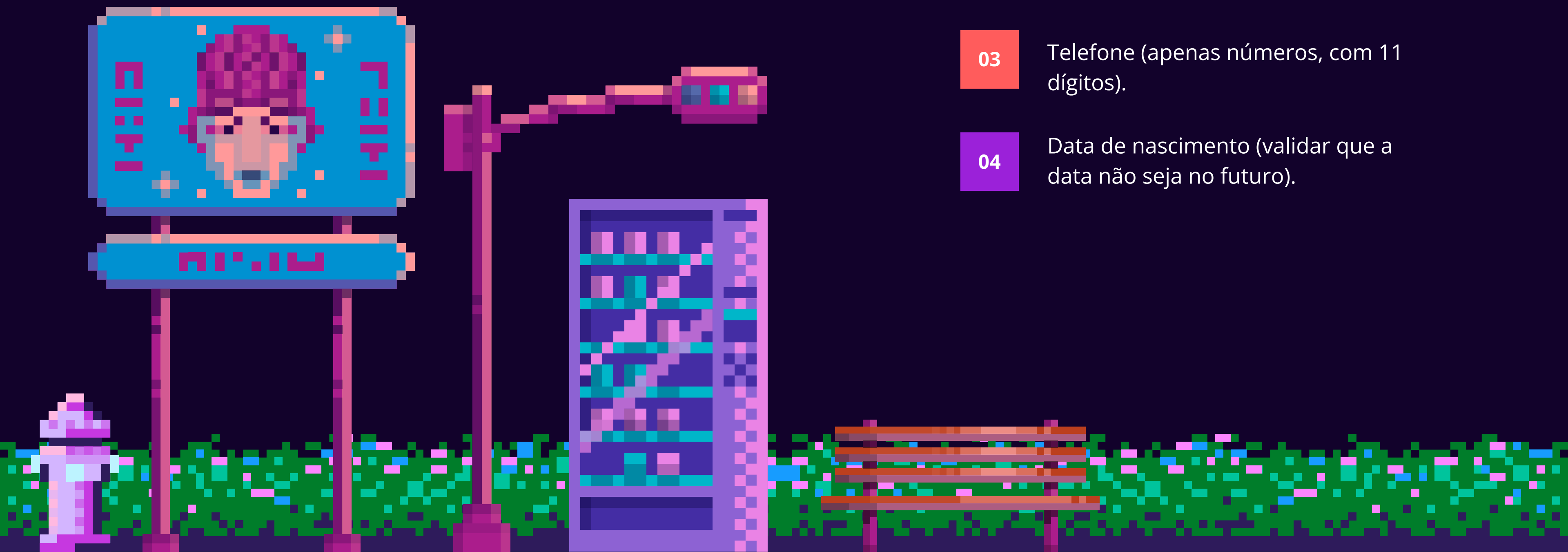
E-mail (precisa ter “@”, um email válido, e “.com”)

03

Telefone (apenas números, com 11 dígitos).

04

Data de nascimento (validar que a data não seja no futuro).





THANK  
YOU