

Conteúdos e projetos

HTML

O HTML (Hypertext Markup Language) é a linguagem padrão para criar páginas web. Abaixo está uma estrutura básica de um documento HTML:

Estrutura de html :

```
<!DOCTYPE html>

<html lang="pt-br">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Título da Página</title>

  <!-- Inclua aqui links para folhas de estilo CSS, scripts JavaScript, etc. -->

</head>

<body>

  <!-- Conteúdo da página -->

</body>

</html>
```

Tags essenciais :

Existem várias tags essenciais em HTML que são comumente utilizadas para estruturar o conteúdo de uma página web. Aqui estão algumas das tags mais essenciais:

<!DOCTYPE html>: Declara o tipo de documento HTML5.

<html>: Elemento raiz do documento.

<head>: Contém informações sobre o documento, como meta tags, título, e links para folhas de estilo e scripts.

<title>: Define o título da página, exibido na barra de título do navegador.

<meta charset="UTF-8">: Define a codificação de caracteres como UTF-8.

<meta name="viewport" content="width=device-width, initial-scale=1.0">: Configura a visualização para dispositivos móveis.

<body>: Contém o conteúdo da página.

<h1>, <h2>, ..., <h6>: Cabeçalhos de diferentes níveis, usados para estruturar títulos e subtítulos.

<p>: Define um parágrafo.

: Cria um link. O atributo href especifica o destino do link.

: Inclui uma imagem na página.

 e : Listas não ordenadas e ordenadas, respectivamente.

: Elemento da lista.

<div>: Usado para agrupar elementos e aplicar estilos.

: Usado para aplicar estilos a partes específicas do texto.

: Insere uma quebra de linha.

<hr>: Cria uma linha horizontal.

 e : Enfatiza o texto, itálico e negrito, respectivamente.

<a>: Define um hiperlink.

<form>: Define um formulário para entrada de dados.

<input>: Define um campo de entrada em um formulário.

<table>: Define a tabela.

<tr>: Define uma linha na tabela.

<th>: Define um cabeçalho de coluna. Normalmente usado dentro da primeira <tr>.

<td>: Define uma célula na tabela. Usado para os dados dentro da tabela.

CSS

O CSS (Cascading Style Sheets) é uma linguagem usada para estilizar documentos HTML. Aqui estão alguns comandos CSS básicos que você pode usar para estilizar elementos HTML, incluindo aqueles em uma tabela:

Seletores:

.classe { propriedade: valor; }: Aplica um estilo a elementos com uma determinada classe.

#id { propriedade: valor; }: Aplica um estilo a um elemento com um ID específico.

elemento { propriedade: valor; }: Aplica um estilo diretamente a um tipo de elemento.

Estilo de Texto:

```
body {  
    font-family: "Arial", sans-serif;  
    color: #333;  
}
```

```
h2 {  
    text-align: center;  
}
```

Estilo de Tabela:

```
table {  
    width: 100%;  
    border-collapse: collapse;  
    margin-top: 20px;  
}
```

```
th, td {  
    border: 1px solid #ddd;  
    padding: 8px;  
    text-align: left;  
}
```

```
th {  
    background-color: #f2f2f2;  
}
```

Estilo de Links:

```
a {  
    text-decoration: none;
```

```
color: #0077cc;  
}
```

```
a:hover {  
    text-decoration: underline;  
}
```

Estilo de Fundo:

```
body {  
    background-color: #f8f8f8;  
}
```

JAVASCRIPT

Comandos Básicos:

1. Variáveis e Tipos de Dados:

```
// Declaração de variáveis
```

```
let nome = "João";
```

```
const idade = 25;
```

```
// Tipos de dados
```

```
let numero = 42; // number
```

```
let texto = "Olá, mundo!"; // string
```

```
let booleano = true; // boolean
```

2. Operadores:

```
let soma = 5 + 3;
```

```
let subtracao = 10 - 4;
```

```
let multiplicacao = 6 * 7;
```

```
let divisao = 20 / 2;  
let resto = 15 % 4; // Resto da divisão
```

3. Estruturas de Controle:

```
// if-else  
let temperatura = 25;  
if (temperatura > 30) {  
    console.log("Está quente!");  
} else {  
    console.log("Está agradável.");  
}  
  
// switch  
let diaDaSemana = "Segunda";  
switch (diaDaSemana) {  
    case "Segunda":  
        console.log("Dia útil");  
        break;  
    case "Sábado":  
    case "Domingo":  
        console.log("Fim de semana");  
        break;  
    default:  
        console.log("Outro dia");  
}
```

4. Laços de Repetição:

```
// for  
for (let i = 0; i < 5; i++) {  
    console.log(i);  
}
```

```
// while
let contador = 0;
while (contador < 3) {
  console.log(contador);
  contador++;
}
```

5. Funções:

```
// Declaração de função
function saudacao(nome) {
  return "Olá, " + nome + "!";
}

// Chamada de função
let mensagem = saudacao("Maria");
console.log(mensagem);
```

Conceitos Essenciais:

1. Objetos:

```
// Definindo um objeto
let pessoa = {
  nome: "Carlos",
  idade: 30,
  cidade: "São Paulo"
};

// Acessando propriedades
console.log(pessoa.nome); // Carlos
```

2. Arrays:

```
// Criando um array  
let frutas = ["Maçã", "Banana", "Morango"];  
  
// Acessando elementos  
console.log(frutas[0]); // Maçã
```

3. Manipulação do DOM:

```
// Selecionando elementos  
let elemento = document.getElementById("idDoElemento");  
  
// Manipulando conteúdo  
elemento.innerHTML = "Novo conteúdo";  
  
// Adicionando/removendo classes  
elemento.classList.add("nova-classe");  
elemento.classList.remove("classe-antiga");
```

4. Eventos:

```
// Adicionando um ouvinte de evento  
elemento.addEventListener("click", function() {  
    alert("Clicou no elemento!");  
});
```

5. Requisições Assíncronas (AJAX):

```
// Exemplo usando o objeto XMLHttpRequest  
let xhr = new XMLHttpRequest();  
xhr.open("GET", "https://api.exemplo.com/dados", true);  
xhr.onreadystatechange = function() {  
    if (xhr.readyState == 4 && xhr.status == 200) {  
        let resposta = JSON.parse(xhr.responseText);
```

```
        console.log(resposta);  
    }  
};  
xhr.send();
```

Comandos Básicos:

1. Variáveis e Tipos de Dados:

- `let nome = "João";` e `const idade = 25;`: Declaração de variáveis usando `let` e `const`. `let` permite reatribuição, enquanto `const` cria uma variável com valor constante.
- Tipos de dados como `number`, `string`, e `boolean` são exemplos de tipos primitivos em JavaScript.

2. Operadores:

- `+`, `-`, `*`, `/`: Operadores aritméticos para adição, subtração, multiplicação e divisão.
- `%`: Operador módulo, que retorna o resto da divisão.

3. Estruturas de Controle:

- `if-else`: Condição que executa um bloco de código se a condição for verdadeira e outro bloco se for falsa.
- `switch`: Estrutura que avalia uma expressão, comparando-a com casos possíveis.

4. Laços de Repetição:

- `for`: Laço de repetição que executa um bloco de código um número específico de vezes.
- `while`: Laço de repetição que executa um bloco de código enquanto uma condição for verdadeira.

5. Funções:

- `function saudacao(nome) { return "Olá, " + nome + "!"; }`: Declaração de uma função chamada `saudacao` que aceita um parâmetro `nome` e retorna uma string de saudação.
- `saudacao("Maria")`: Chamada da função com um argumento.

Conceitos Essenciais:

1. Objetos:

- `let pessoa = { nome: "Carlos", idade: 30, cidade: "São Paulo" };`: Declaração de um objeto chamado `pessoa` com propriedades `nome`, `idade` e `cidade`.
- `pessoa.nome`: Acesso a uma propriedade do objeto.

2. Arrays:

- `let frutas = ["Maçã", "Banana", "Morango"];`: Declaração de um array chamado `frutas`.
- `frutas[0]`: Acesso a um elemento do array.

3. Manipulação do DOM:

- `document.getElementById("idDoElemento")`: Seleção de um elemento no DOM por seu ID.
- `elemento.innerHTML = "Novo conteúdo";`: Manipulação do conteúdo HTML de um elemento.
- `elemento.classList.add("nova-classe")`: Adição de uma classe a um elemento.

4. Eventos:

- `elemento.addEventListener("click", function() { alert("Clicou no elemento!"); });`: Adição de um ouvinte de evento que responde a um clique no elemento.

5. Requisições Assíncronas (AJAX):

- Uso do objeto `XMLHttpRequest` para fazer requisições assíncronas a servidores web.
- `xhr.onreadystatechange`: Manipulação do estado da requisição para garantir que ela foi concluída com sucesso.

Esses são conceitos essenciais para começar com JavaScript, mas há muito mais a explorar à medida que você se aprofunda na linguagem e suas aplicações práticas.

PYTHON

Comandos Básicos:

1. Variáveis e Tipos de Dados:

```
nome = "João"
```

```
idade = 25
```

```
numero = 42
```

```
texto = "Olá, mundo!"
```

```
booleano = True
```

- `nome` e `idade`: Declaração de variáveis. `nome` é uma string, e `idade` é um número inteiro.

- `numero`, `texto` e `booleano`: Exemplos de diferentes tipos de dados em Python, incluindo inteiros, strings e booleanos.

2. Operadores:

```
soma = 5 + 3
```

```
subtracao = 10 - 4
```

```
multiplicacao = 6 * 7
```

```
divisao = 20 / 2
```

```
resto = 15 % 4
```

- Operadores aritméticos básicos: `+` (adição), `-` (subtração), `*` (multiplicação), `/` (divisão), `%` (resto da divisão).

3. Estruturas de Controle:

```
temperatura = 25
```

```
if temperatura > 30:
```

```
    print("Está quente!")
```

```
else:
```

```
    print("Está agradável.")
```

- ``if-else``: Estrutura de controle de fluxo condicional. Se a temperatura for maior que 30, imprime "Está quente!", caso contrário, imprime "Está agradável."

4. Laços de Repetição:

```
for i in range(5):
```

```
    print(i)
```

- ``for``: Laço de repetição usado para iterar sobre uma sequência (neste caso, ``range(5)``).

```
contador = 0
```

```
while contador < 3:
```

```
    print(contador)
```

```
    contador += 1
```

- ``while``: Laço de repetição que executa um bloco de código enquanto uma condição é verdadeira.

5. Funções:

```
def saudacao(nome):
```

```
    return "Olá, " + nome + "!"
```

```
mensagem = saudacao("Maria")
```

```
print(mensagem)
```

- ``def saudacao(nome)``: Declaração de uma função chamada ``saudacao`` que aceita um parâmetro ``nome``.

- ``return "Olá, " + nome + "!"``: A função retorna uma string de saudação.

- ``mensagem = saudacao("Maria")``: Chamada da função com um argumento "Maria".

- ``print(mensagem)``: Exibe a saudação no console.

Conceitos Essenciais:

1. Listas:

```
frutas = ["Maçã", "Banana", "Morango"]
```

```
print(frutas[0]) # Maçã
```

- ``frutas``: Declaração de uma lista contendo strings.
- ``frutas[0]``: Acessando o primeiro elemento da lista (índice 0).

2. Dicionários:

```
peessoa = {"nome": "Carlos", "idade": 30, "cidade": "São Paulo"}  
print(peessoa["nome"]) # Carlos
```

- `peessoa``: Declaração de um dicionário com chaves e valores.
- ``peessoa["nome"]``: Acessando o valor associado à chave "nome" no dicionário.

3. Compreensões de Lista:

```
numeros_pares = [x for x in range(10) if x % 2 == 0]
```

- Compreensão de lista para criar uma lista de números pares de 0 a 9.

4. Tratamento de Exceções:

```
try:  
    resultado = 10 / 0  
except ZeroDivisionError as e:  
    print(f"Erro: {e}")
```

- Uso de blocos ``try`` e ``except`` para tratar exceções. No exemplo, evitando a divisão por zero.

5. Importação de Módulos:

```
import math  
raiz_quadrada = math.sqrt(25)
```

- ``import math``: Importação do módulo ``math``.
- ``math.sqrt(25)``: Uso da função ``sqrt`` do módulo ``math`` para calcular a raiz quadrada de 25.

Esses são os detalhes dos comandos e conceitos essenciais em Python. Cada um desempenha um papel fundamental na escrita de código eficiente e compreensível em Python.

FLASK

Flask:

Flask é um framework web para Python, projetado para ser simples, leve e fácil de usar. Ele permite criar aplicativos web de forma rápida e eficiente. Vamos detalhar alguns comandos e conceitos fundamentais do Flask:

Comandos Básicos:

1. Instalação do Flask:

```
pip install Flask
```

- Comando para instalar o Flask usando o gerenciador de pacotes `pip`.

2. Estrutura Básica de um Aplicativo Flask:

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')
```

```
def hello_world():
```

```
    return 'Hello, World!'
```

```
if __name__ == '__main__':
```

```
    app.run()
```

```
...
```

- `from flask import Flask`: Importa a classe `Flask` do módulo Flask.

- `app = Flask(__name__)`: Cria uma instância do aplicativo Flask.

- `@app.route('/')`: Define uma rota para a função `hello_world`.

- `def hello_world()`: Função que retorna "Hello, World!".

- `app.run()`: Inicia o servidor web embutido quando o script é executado diretamente.

3. Executando o Aplicativo:

```
python nome_do_app.py
```

- Comando para iniciar o servidor Flask.

Conceitos Essenciais:

1. Rotas:

```
@app.route('/pagina')
```

```
def pagina():
```

```
    return 'Conteúdo da Página'
```

- Decorador `@app.route('/pagina')`: Define uma rota para a função `pagina`.
- A função `pagina` é chamada quando a rota é acessada.

2. Templates:

```
from flask import render_template
```

```
@app.route('/template')
```

```
def pagina_template():
```

```
    return render_template('template.html', variavel='Valor')
```

- `render_template`: Função para renderizar templates.
- Pode passar variáveis para o template, como `variavel='Valor'`.

3. Requisições e Parâmetros:

```
from flask import request
```

```
@app.route('/parametros')
```

```
def parametros():
```

```
    nome = request.args.get('nome', 'Visitante')
```

```
    return f'Olá, {nome}!'
```

- `request.args.get('nome', 'Visitante')`: Obtém o parâmetro 'nome' da requisição ou usa 'Visitante' como padrão.

4. Formulários:

```
from flask import request
```

```
@app.route('/formulario', methods=['POST'])
def formulario():
    nome = request.form.get('nome', 'Visitante')
    return f'Olá, {nome}!'
...
```

- `methods=['POST']`: Especifica que a rota só aceita requisições POST.

- `request.form.get('nome', 'Visitante')`: Obtém dados de formulário enviado via POST.

5. Redirecionamento:

```
from flask import redirect
```

```
@app.route('/redirecionar')
def redirecionar():
    return redirect('/nova_rota')
...
```

- `redirect('/nova_rota')`: Redireciona para a rota '/nova_rota'.

Para explorar mais recursos: [Documentação do Flask](https://flask.palletsprojects.com/).

SQL

SQL (Structured Query Language):

SQL é uma linguagem de consulta estruturada usada para gerenciar e manipular bancos de dados relacionais. Aqui estão alguns comandos e conceitos fundamentais em SQL:

Comandos Básicos:

1. Criação de Tabela:

```
CREATE TABLE usuarios (  
    id INT PRIMARY KEY,  
    nome VARCHAR(50),  
    idade INT  
);
```

- CREATE TABLE: Comando para criar uma tabela no banco de dados.
- usuarios: Nome da tabela.
- `id`, `nome`, `idade`: Nomes das colunas e seus tipos.

2. Inserção de Dados:

```
INSERT INTO usuarios (id, nome, idade) VALUES (1, 'João', 25);
```

- `INSERT INTO`: Comando para inserir dados em uma tabela.
- usuarios: Nome da tabela.
- (id, nome, idade): Nomes das colunas.
- VALUES (1, 'João', 25): Valores a serem inseridos.

3. Consulta de Dados:

```
SELECT nome, idade FROM usuarios WHERE idade > 20;
```

- SELECT: Comando para selecionar dados de uma tabela.
- nome, idade: Colunas a serem selecionadas.
- FROM usuarios: Nome da tabela.
- WHERE idade > 20: Condição para filtrar os resultados.

4. Atualização de Dados:

```
UPDATE usuarios SET idade = 26 WHERE nome = 'João';
```

- UPDATE: Comando para atualizar dados em uma tabela.
- usuarios: Nome da tabela.
- SET idade = 26: Atualiza o valor da coluna `idade`.
- WHERE nome = 'João': Condição para identificar a linha a ser atualizada.

5. Exclusão de Dados:

```
DELETE FROM usuarios WHERE nome = 'João';
```

- DELETE FROM: Comando para excluir dados de uma tabela.

- usuarios: Nome da tabela.

- `WHERE nome = 'João'`: Condição para identificar a linha a ser excluída.

Conceitos Essenciais:

1. Chave Primária:

```
CREATE TABLE produtos (  
    id INT PRIMARY KEY,  
    nome VARCHAR(50),  
    preco DECIMAL(10, 2)  
);
```

- `PRIMARY KEY`: Define a coluna `id` como chave primária.

2. Relacionamentos:

```
CREATE TABLE pedidos (  
    id INT PRIMARY KEY,  
    produto_id INT,  
    quantidade INT,  
    FOREIGN KEY (produto_id) REFERENCES produtos(id)  
);
```

- `FOREIGN KEY (produto_id) REFERENCES produtos(id)`: Estabelece uma chave estrangeira referenciando a tabela `produtos`.

3. Índices:

```
CREATE INDEX idx_nome ON usuarios (nome);
```

- `CREATE INDEX`: Cria um índice para acelerar consultas na coluna `nome`.

4. Agregação:

```
SELECT COUNT(*) FROM usuarios;
```

- COUNT(*): Função de agregação que conta o número total de registros.

5. Agrupamento:

```
SELECT cidade, COUNT(*) FROM clientes GROUP BY cidade;
```

- GROUP BY cidade: Agrupa os resultados pela coluna `cidade`.

Estes são alguns dos comandos e conceitos fundamentais em SQL. SQL é uma linguagem poderosa usada em uma variedade de sistemas de gerenciamento de bancos de dados relacionais, como MySQL, PostgreSQL, SQLite e Oracle. Cada sistema pode ter variações em sua implementação específica. Consulte a documentação do sistema de gerenciamento de banco de dados que você está utilizando para obter informações detalhadas.

GIT/GITHUB

Git é um sistema de controle de versão distribuído, e GitHub é uma plataforma de hospedagem de código que utiliza o Git para controle de versão. Aqui estão alguns comandos e conceitos fundamentais em Git e GitHub:

Comandos Básicos do Git:

1. Iniciar um Repositório:

```
git init
```

- `git init`: Inicia um novo repositório Git no diretório atual.

2. Clonar um Repositório Existente:

```
git clone url_do_repositorio
```

- git clone: Clona um repositório Git existente para o seu sistema local.

3. Adicionar Mudanças ao Stage:

```
git add arquivo_modificado
```

- git add: Adiciona mudanças ao stage para serem preparadas para o commit.

4. ****Comitar Mudanças:****

`git commit -m "Mensagem do Commit"`

- `git commit`: Registra as mudanças no repositório com uma mensagem descritiva.

5. Verificar Status:

`git status`

- `git status`: Exibe o status das mudanças (não commitadas, no stage, etc.).

6. Atualizar do Repositório Remoto:

`git pull origin nome_da_branch`

- `git pull`: Atualiza o repositório local com as alterações do repositório remoto.

7. Enviar Mudanças para o Repositório Remoto:

`git push origin nome_da_branch`

- `git push`: Envia mudanças locais para o repositório remoto.

Conceitos Essenciais do Git/GitHub:

1. Branches:

`git branch nome_da_branch`

`git checkout nome_da_branch`

- `git branch`: Cria uma nova branch.

- `git checkout`: Muda para uma branch existente.

2. Merge:

`git merge nome_da_branch`

- `git merge`: Mescla as alterações de uma branch para outra.

3. Conflitos de Merge:

- Durante o merge, se houver conflitos, você precisará resolvê-los manualmente.

4. Pull Requests (GitHub):

- No GitHub, um Pull Request (PR) é uma proposta de alteração que você envia para o proprietário do repositório para revisão.

5. Forks e Clones (GitHub):

- Um fork é uma cópia de um repositório. Clonar um fork cria uma cópia local no seu sistema.

6. Issues (GitHub):

- As issues são usadas para rastrear tarefas, bugs ou discussões relacionadas ao projeto.

7. Gitignore:

- O arquivo `.gitignore` especifica quais arquivos ou diretórios devem ser ignorados pelo Git.

8. Revert:

`git revert hash_do_commit`

- ``git revert``: Desfaz as alterações introduzidas por um commit específico, criando um novo commit.

Estes são alguns dos comandos e conceitos fundamentais em Git e GitHub. O Git é uma ferramenta poderosa para o controle de versão, enquanto o GitHub facilita a colaboração e o compartilhamento de código entre desenvolvedores. Familiarizar-se com esses conceitos é essencial para trabalhar eficientemente em projetos de desenvolvimento de software.

DEPLOY-STATIC

Vercel:

[Vercel](https://vercel.com/) é uma plataforma de implantação (deploy) e hospedagem que simplifica o processo para aplicativos da web, especialmente para sites estáticos e projetos baseados em React, Next.js, entre outros. Aqui estão os passos básicos:

1. Criar uma Conta:

- Você precisa se inscrever para uma conta no Vercel.

2. Configurar um Projeto:

- Vincule seu repositório Git ao Vercel.

3. Configuração Automática:

- O Vercel detectará automaticamente as configurações do seu projeto e criará as implantações necessárias.

4. Domínios Personalizados e Configurações Avançadas:

- O Vercel facilita a configuração de domínios personalizados, certificados SSL e outras configurações avançadas.

5. Deploy Contínuo:

- Cada vez que você faz push no seu repositório, o Vercel pode ser configurado para implantar automaticamente as alterações.

Render:

[Render](https://render.com/) é uma plataforma de nuvem que oferece hospedagem fácil para aplicativos web e APIs. Aqui estão alguns passos básicos:

1. Criar uma Conta:

- Inscreva-se para uma conta Render.

2. Criar um Novo Serviço:

- Adicione um novo serviço ao Render e selecione o tipo de serviço (por exemplo, site estático, aplicativo da web, etc.).

3. Configurar Implantação:

- Conecte seu repositório Git ao Render e configure as opções de implantação.

4. Ambientes e Variáveis de Ambiente:

- Render suporta ambientes e variáveis de ambiente, permitindo a configuração flexível de suas implantações.

5. Domínios Personalizados e Certificados SSL:

- Configure domínios personalizados e Render pode gerenciar automaticamente os certificados SSL.

Ambos Vercel e Render são conhecidos por sua facilidade de uso, integração direta com repositórios Git, e automatização do processo de deploy. A escolha entre eles pode depender dos requisitos específicos do seu projeto, das características desejadas e das preferências pessoais.

Projetos:

Vamos criar um projeto simples que envolve todas essas tecnologias. Vamos criar um site estático que exibe informações de usuários a partir de um banco de dados SQL usando Flask como backend, e usaremos Git/GitHub para controle de versão e Vercel para deploy estático.

Projeto: Site de Perfil de Usuário

HTML (index.html):

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<link rel="stylesheet" href="style.css">

<title>User Profile</title>

</head>

<body>

  <div id="app">

    <h1>User Profile</h1>

    <div id="user-info"></div>

  </div>

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

</body>

</html>
```

CSS (style.css):

```
body {

  font-family: Arial, sans-serif;

  margin: 0;

  padding: 0;

  background-color: #f4f4f4;

}


#app {

  text-align: center;

  margin: 50px;

}


#user-info {

  background-color: #fff;

  padding: 20px;

  border-radius: 8px;

  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}
```

JavaScript (script.js):

```
document.addEventListener("DOMContentLoaded", () => {
    fetchUserData();
});
```

```
async function fetchUserData() {
    try {
        const response = await fetch("/api/user");
        const userData = await response.json();
        displayUserData(userData);
    } catch (error) {
        console.error("Error fetching user data:", error);
    }
}
```

```
function displayUserData(userData) {
    const userInfoDiv = document.getElementById("user-info");
    userInfoDiv.innerHTML = `<h2>${userData.name}</h2>
        <p>Email: ${userData.email}</p>
        <p>Age: ${userData.age}</p>`;
}
```

Python (app.py - Flask com MySQL):

```
from flask import Flask, jsonify
import mysql.connector
```

```
app = Flask(__name__)
```

```
# Configurações do banco de dados
```

```
db_config = {
    'host': 'seu_host_mysql',
    'user': 'seu_usuario_mysql',
    'password': 'sua_senha_mysql',
```



```
'database': 'seu_banco_de_dados_mysql'
}
```

Endpoint para obter dados do usuário

```
@app.route('/api/user')
```

```
def get_user_data():
```

```
    connection = mysql.connector.connect(**db_config)
```

```
    cursor = connection.cursor()
```

Criação da tabela se não existir

```
cursor.execute("""CREATE TABLE IF NOT EXISTS users (
```

```
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    name VARCHAR(255),
```

```
    email VARCHAR(255),
```

```
    age INT)""")
```

```
connection.commit()
```

Inserção de dados fictícios

```
cursor.execute("INSERT INTO users (name, email, age) VALUES (%s, %s, %s)", ('John Doe',
'john@example.com', 25))
```

```
connection.commit()
```

Busca dos dados na tabela

```
cursor.execute("SELECT * FROM users LIMIT 1")
```

```
user_data = cursor.fetchone()
```

```
connection.close()
```

```
return jsonify({
```

```
    'name': user_data[1],
```

```
    'email': user_data[2],
```

```
        'age': user_data[3]  
    })
```

```
if __name__ == '__main__':
```

```
    app.run(debug=True)
```

Certifique-se de substituir 'seu_host_mysql', 'seu_usuario_mysql', 'sua_senha_mysql' e 'seu_banco_de_dados_mysql' pelos detalhes de conexão corretos para o seu ambiente MySQL.

Além disso, você precisará instalar o módulo mysql-connector-python. Você pode fazer isso executando o seguinte comando no seu terminal:

```
pip install mysql-connector-python
```

Lembre-se de criar um banco de dados no MySQL e fornecer as credenciais corretas no arquivo db_config. Este exemplo é bastante básico e serve como uma introdução para interagir com o MySQL usando Python e Flask. Certifique-se de adaptar o código conforme necessário para atender aos requisitos específicos do seu projeto. [Git/GitHub](#):

Para criar um arquivo SQL que define a estrutura do banco de dados MySQL, você pode criar um arquivo com extensão .sql. Vou fornecer um exemplo simples para a tabela de usuários mencionada no projeto anterior:

Arquivo SQL (create_database.sql):

```
-- Criação do banco de dados
```

```
CREATE DATABASE IF NOT EXISTS `seu_banco_de_dados_mysql`;
```

```
USE `seu_banco_de_dados_mysql`;
```

```
-- Criação da tabela de usuários
```

```
CREATE TABLE IF NOT EXISTS `users` (
```

```
    `id` INT AUTO_INCREMENT PRIMARY KEY,
```

```
    `name` VARCHAR(255) NOT NULL,
```

```
    `email` VARCHAR(255) NOT NULL,
```

```
    `age` INT
```

```
);
```

Substitua 'seu_banco_de_dados_mysql' pelo nome desejado para o seu banco de dados. Este script cria o banco de dados (se não existir) e a tabela users com as colunas id, name, email e age.

Depois de criar este script, você pode executá-lo em seu cliente MySQL ou usando a linha de comando:

```
mysql -u seu_usuario_mysql -p < create_database.sql
```

Isso solicitará sua senha do MySQL e executará as instruções SQL no arquivo para criar o banco de dados e a tabela.

Certifique-se de ter o MySQL instalado e acessível no seu sistema. Adapte o script conforme necessário para atender às necessidades específicas do seu projeto.

Inicialize um repositório no diretório do seu projeto:

```
git init
```

Adicione os arquivos ao stage e faça o primeiro commit:

```
git add .
```

```
git commit -m "Initial commit"
```

Crie um repositório no GitHub e adicione o remote:

```
git remote add origin <URL_DO_REPOSITORIO_NO_GITHUB>
```

Envie o código para o GitHub:

```
git push -u origin master
```

Deploy Estático (Vercel):

Crie uma conta no Vercel.

Adicione um novo projeto usando seu repositório do GitHub.

O Vercel detectará automaticamente as configurações e implantará o projeto.

Este é um exemplo básico para ilustrar como você pode integrar HTML, CSS, JS, Python (Flask), SQL, Git/GitHub e deploy estático usando Vercel.

Jogo da forca

(python)

Segue abaixo um os primeiros passos para o jogo da forca em python:

1-Escolher uma palavra aleatória:

Para escolher uma palavra aleatória do conjunto de palavras disponíveis, podemos utilizar a biblioteca random do Python. Armazenamos todas as palavras possíveis em uma lista e utilizamos a função random.choice() para selecionar uma palavra aleatória dessa lista.

2-Armazenar a palavra secreta:

Podemos armazenar a palavra secreta em uma variável. Para isso, depois de escolher a palavra aleatória, podemos definir a variável palavra_secreta para armazenar essa palavra.

3-Exibir a palavra oculta e as letras corretamente adivinhadas:

Podemos criar uma função que exiba a palavra oculta com traços ou underscores (_) representando as letras não adivinhadas. À medida que o jogador acerta letras, esses traços serão substituídos pelas letras corretas. Podemos utilizar laços de repetição para percorrer cada letra da palavra e verificar se ela foi adivinhada corretamente.

4-Receber a entrada do jogador:

Podemos utilizar a função input() do Python para receber a entrada do jogador, isto é, as letras que ele acredita que estão na palavra. Essas letras são armazenadas em uma lista ou conjunto para evitar repetições.

5-Verificar a entrada do jogador:

Devemos verificar se a letra inserida pelo jogador está presente na palavra secreta. Se estiver, a letra é adicionada à lista de letras corretamente adivinhadas. Se não estiver, o jogador perde uma tentativa.

6-Controlar as tentativas:

Podemos utilizar uma variável para acompanhar o número de tentativas restantes. A cada vez que o jogador inserir uma letra errada, subtraímos 1 dessa variável. Quando o número de tentativas chegar a 0, o jogador perde o jogo.

7-Exibir a forca:

Podemos criar uma função que exiba a figura da forca, representando as tentativas perdidas pelo jogador. Cada vez que o jogador erra uma letra, uma parte do boneco é desenhada.

8-Verificar a vitória ou derrota:

A cada tentativa, verificamos se o jogador adivinhou todas as letras corretas da palavra. Se todas as letras foram adivinhadas, o jogador vence. Se o número de tentativas chegar a 0 antes que todas as letras sejam adivinhadas, o jogador perde.

9-Loop do jogo:

O jogo deve ser executado em um loop até que o jogador vença ou perca. Após cada tentativa do jogador, verificamos a condição de vitória ou derrota. Se o jogo acabar, perguntamos ao jogador se ele deseja jogar novamente.

Codigo

```
import random
```

```
def escolher_palavra():
```

```
    palavras = ["python", "programacao", "computador", "jogo", "algoritmo", "inteligencia"]
```

```
    return random.choice(palavras)
```

```
def mostrar_forca(palavra, letras_corretas):
```

```
    for letra in palavra:
```

```
        if letra in letras_corretas:
```

```
            print(letra, end=" ")
```

```
        else:
```

```
            print("_", end=" ")
```

```
    print()
```

```
def jogar_forca():
```

```
    palavra_secreta = escolher_palavra()
```

```
    letras_corretas = []
```

```
    tentativas = 6
```

```
print("Bem-vindo ao Jogo da Forca!")

mostrar_forca(palavra_secreta, letras_corretas)

while tentativas > 0:

    letra = input("Digite uma letra: ")

    if letra in letras_corretas:

        print("Você já tentou essa letra.")

    elif letra in palavra_secreta:

        letras_corretas.append(letra)

    else:

        print("Letra errada!")

        tentativas -= 1

    mostrar_forca(palavra_secreta, letras_corretas)

    if all(letra in letras_corretas for letra in palavra_secreta):

        print("Parabéns! Você acertou a palavra!")

        break

if tentativas == 0:

    print("Você perdeu! A palavra secreta era:", palavra_secreta)

if __name__ == "__main__":

    jogar_forca()
```

Explicação do código

1. ``import random``: Esta linha importa o módulo ``random``, que será usado para escolher uma palavra aleatória da lista.

2. ``def escolher_palavra():``: Define uma função chamada ``escolher_palavra()`` que não recebe argumentos. Essa função retorna uma palavra aleatória escolhida da lista ``palavras``.

3. ``palavras = ["python", "programacao", "computador", "jogo", "algoritmo", "inteligencia"]``: Uma lista de palavras é criada com algumas opções para o jogo da forca.

4. ``return random.choice(palavras)``: A função ``escolher_palavra()`` usa o ``random.choice()`` para retornar uma palavra aleatória da lista ``palavras``.

5. ``def mostrar_forca(palavra, letras_corretas):``: Define a função ``mostrar_forca(palavra, letras_corretas)``, que recebe duas entradas: a palavra secreta (``palavra``) e uma lista de letras corretas já informadas pelo jogador (``letras_corretas``).

6. ``for letra in palavra:``: O loop ``for`` é usado para iterar por cada letra na ``palavra``.

7. ``if letra in letras_corretas:``: Verifica se a letra atual já foi informada e está presente na lista de ``letras_corretas``.

8. ``print(letra, end=" ")``: Se a letra estiver correta, é mostrada na tela, seguida de um espaço. Caso contrário, um ``_`` é mostrado no lugar da letra.

9. ``print()``: Imprime uma nova linha após mostrar todas as letras corretas ou ``_``.

10. ``def jogar_forca():``: Define a função ``jogar_forca()`` que não recebe argumentos.

11. ``palavra_secreta = escolher_palavra()``: Chama a função ``escolher_palavra()`` para obter

uma palavra aleatória e armazena em ``palavra_secreta``.

12. ``letras_corretas = []``: Inicializa uma lista vazia para armazenar as letras corretas informadas pelo jogador.

13. ``tentativas = 6``: Inicializa o número de tentativas disponíveis para o jogador.

14. ``print("Bem-vindo ao Jogo da Forca!")``: Imprime uma mensagem de boas-vindas.

15. ``mostrar_forca(palavra_secreta, letras_corretas)``: Chama a função ``mostrar_forca()`` para mostrar o estado atual da palavra, com as letras corretas adivinhadas e os espaços para as letras ainda não adivinhadas.

16. ``while tentativas > 0:``: Entra em um loop que continuará até que o número de ``tentativas`` seja igual a zero.

17. ``letra = input("Digite uma letra: ")``: Solicita ao jogador que digite uma letra.

18. ``if letra in letras_corretas:``: Verifica se a letra já foi tentada anteriormente e está na lista de ``letras_corretas``.

19. ``print("Você já tentou essa letra.")``: Se a letra já tiver sido tentada, imprime uma mensagem informando que ela já foi usada.

20. ``elif letra in palavra_secreta:``: Verifica se a letra está na ``palavra_secreta``.

21. ``letras_corretas.append(letra)``: Se a letra estiver correta e não tiver sido tentada antes, ela é adicionada à lista de ``letras_corretas``.

22. ``else:``: Se a letra não estiver correta e não tiver sido tentada antes, ela é considerada uma tentativa inválida.

23. ``print("Letra errada!")``: Imprime uma mensagem informando que a letra é incorreta.
24. ``tentativas -= 1``: Diminui o número de ``tentativas`` em 1, pois uma tentativa foi usada.
25. ``mostrar_forca(palavra_secreta, letras_corretas)``: Chama a função ``mostrar_forca()`` novamente para atualizar o estado da palavra com as letras corretas adivinhadas e os espaços para as letras não adivinhadas.
26. ``if all(letra in letras_corretas for letra in palavra_secreta):``: Verifica se todas as letras da ``palavra_secreta`` estão na lista ``letras_corretas``, o que significa que o jogador adivinhou corretamente todas as letras.
27. ``print("Parabéns! Você acertou a palavra!")``: Se todas as letras foram adivinhadas corretamente, o jogador venceu o jogo e uma mensagem de parabéns é exibida.
28. ``break``: Encerra o loop while, pois o jogador já venceu o jogo.
29. ``if tentativas == 0:``: Se o loop while terminar e o jogador não tiver vencido, verifica se o número de ``tentativas`` chegou a zero.
30. ``print("Você perdeu! A palavra secreta era:", palavra_secreta)``: Se o jogador esgotar todas as tentativas, imprime uma mensagem de derrota e mostra a ``palavra_secreta``.
31. ``if __name__ == "__main__":``: Verifica se o código está sendo executado diretamente (não sendo importado como um módulo).
32. ``jogar_forca()``: Chama a função ``jogar_forca()`` para iniciar o jogo da forca.

Esse código implementa o clássico jogo da forca em que o jogador deve adivinhar uma palavra secreta. O jogador pode digitar uma letra por vez e tem um número limitado de tentativas

para acertar a palavra. A cada tentativa incorreta, uma parte do desenho da forca é montada, e o jogo continua até que o jogador adivinhe corretamente a palavra ou esgote todas as tentativas.