Material de apoio - Módulo 4

HTML

A Linguagem de Marcação de Hipertexto (HTML) é uma linguagem de computador que compõe a maior parte das páginas da internet e dos aplicativos online. Um hipertexto é um texto usado para fazer referência a outros textos, enquanto uma linguagem de marcação é composta por uma série de marcações que dizem para os servidores da web qual é o estilo e a estrutura de um documento.

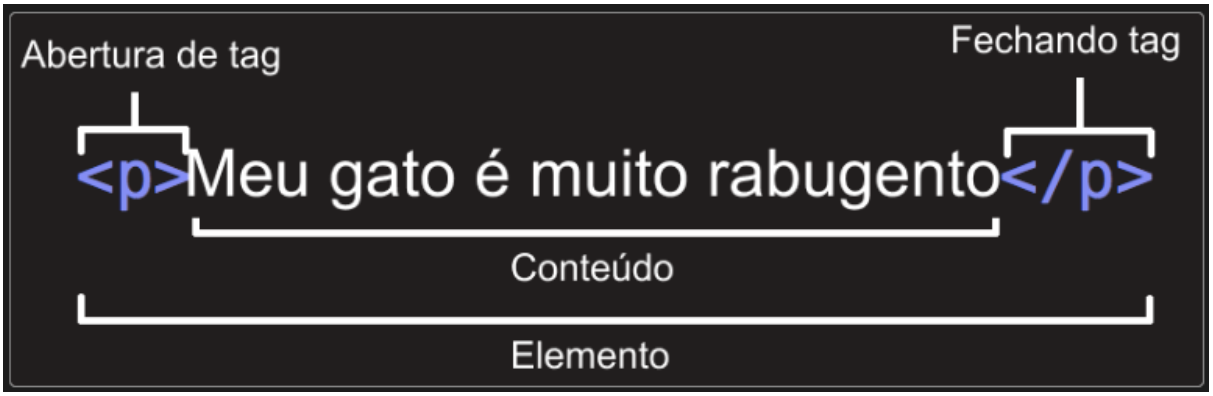
O HTML não é considerado uma linguagem de programação, já que ele não pode criar funcionalidades dinâmicas. Ao invés disso, com o HTML, os usuários podem criar e estruturar seções, parágrafos e links usando elementos, tags e atributos. Quem interpreta essas marcações são o navegador. Ainda, o navegador é bem flexível na questão dos erros, pois ele passa por cima deles e apenas mostra o que for possível na tela ao usuário.

As marcações são feitas através das **tags** no código, sendo que cada tag gera um resultado diferente no texto. As principais tags são:

1. `<!DOCTYPE>`: Esta não é uma tag HTML propriamente dita, mas é usada para definir o tipo de documento e a versão do HTML que a página está usando.
2. `<html>`: Esta é a tag raiz que envolve todo o conteúdo da página. Todas as outras tags HTML são aninhadas dentro desta.
3. `<head>`: Esta tag contém informações sobre a página, como o título, as meta tags (para SEO), links para arquivos externos (como folhas de estilo CSS) e scripts.
4. `<title>`: Usada dentro da tag `<head>`, define o título da página, que é exibido na barra de título do navegador ou na aba.
5. `<meta>`: Usada para fornecer metadados sobre a página, como descrição, palavras-chave e codificação de caracteres.
6. `<link>`: Usada para vincular a página a arquivos externos, como folhas de estilo CSS.
7. `<script>`: Utilizada para incluir scripts JavaScript na página, que podem adicionar interatividade e funcionalidade à página.
8. `<body>`: Contém o conteúdo principal da página, incluindo texto, imagens, links e outros elementos visíveis.

9. `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`: Tags de cabeçalho usadas para criar títulos e subtítulos. `<h1>` é o mais importante e `<h6>` é o menos importante em termos de hierarquia.
10. `<p>`: Usada para criar parágrafos de texto.
11. `<a>`: Usada para criar links para outras páginas da web ou recursos, com um atributo `href` que especifica o destino do link.
12. `<img>`: Usada para incorporar imagens na página, com um atributo `src` que especifica o URL da imagem.
13. `<ul>` e `<ol>`: Usadas para criar listas não ordenadas e ordenadas, respectivamente. As tags `<li>` são usadas para definir itens de lista dentro de `<ul>` ou `<ol>`.
14. `<div>` e `<span>`: Tags de contêiner genéricas usadas para agrupar elementos e aplicar estilos ou scripts a eles.
15. `<table>`, `<tr>`, `<th>`, `<td>`: Usadas para criar tabelas. `<table>` define a tabela, `<tr>` define uma linha, `<th>` define cabeçalhos de coluna e `<td>` define células de dados.
16. `<form>`: Usada para criar formulários interativos que permitem aos usuários inserir e enviar dados.
17. `<input>`, `<textarea>`, `<select>`: Elementos usados dentro de formulários para coletar informações do usuário, como caixas de texto, áreas de texto e menus suspensos.
18. `<iframe>`: Usada para incorporar conteúdo de outras páginas da web dentro de uma página.
19. `<audio>` e `<video>`: Usadas para incorporar áudio e vídeo na página, permitindo a reprodução de mídia diretamente no navegador.
20. `<footer>`, `<header>`, `<nav>`, `<article>`, `<section>`: Tags semânticas introduzidas em HTML5 para melhorar a estrutura e acessibilidade das páginas da web.

HTML consiste de uma série de **elementos**, que você usa para delimitar ou agrupar diferentes partes do conteúdo para que ele apareça ou atue de determinada maneira. As tags anexas podem transformar uma palavra ou imagem num hiperlink, pode colocar palavras em itálico, pode aumentar ou diminuir a fonte e assim por diante. Por exemplo, veja a seguinte linha de conteúdo:



Elementos também podem ter atributos, que parecem assim:

CSS

CSS é a sigla para "Cascading Style Sheets", que em português significa "Folhas de Estilo em Cascata". É uma linguagem de marcação utilizada para definir a apresentação visual de documentos HTML. Em outras palavras, o CSS é usado para controlar o design, o layout e a formatação de páginas da web e documentos estruturados.

O CSS permite que os desenvolvedores web especifiquem como os elementos HTML devem ser exibidos em termos de cores, fontes, margens, espaçamento, posicionamento, tamanhos e muito mais. Isso separa a estrutura e o conteúdo de uma página (geralmente definidos pelo HTML) de sua apresentação visual (definida pelo CSS), tornando mais fácil e flexível o design e a manutenção de sites.

Um exemplo simples de código CSS seria:

```
/* Define a cor do texto para todos os parágrafos */
p {
  color: blue;
}

/* Define o tamanho da fonte para todos os cabeçalhos de nível 1 */
h1 {
  font-size: 24px;
}

/* Define a cor de fundo para todas as divs com a classe "destaque" */
div.destaque {
```

```
  background-color: yellow;
}
```

Nesse exemplo, você pode ver como o CSS é usado para selecionar elementos HTML (como parágrafos, cabeçalhos e elementos com classes específicas) e definir estilos para eles, como cores, tamanhos de fonte e cores de fundo. O CSS permite que os desenvolvedores controlem a aparência e o layout de uma página da web de maneira eficiente e consistente.

Atributos de tags HTML

No HTML, os atributos são valores adicionados a elementos HTML para fornecer informações ou características específicas a esses elementos. Alguns atributos comuns incluem "class", "id", "href", "src", "alt", "style" e muitos outros, dependendo do elemento HTML em questão. Esses atributos são usados para estruturar e enriquecer o conteúdo da página e, em muitos casos, também para referenciar elementos específicos no CSS.

Aqui estão alguns exemplos de como você pode referenciar atributos HTML no CSS:

1. **Atributo "class"**:
   O atributo "class" é usado para agrupar elementos e aplicar estilos a vários elementos com a mesma classe. Para referenciar elementos com uma classe específica no CSS, você usa um ponto (.) seguido do nome da classe:

```
<p class="destaque">Este é um parágrafo de destaque.</p>
```

```
.destaque {
  color: blue;
  font-weight: bold;
}
```

2. **Atributo "id"**:
   O atributo "id" é usado para identificar exclusivamente um elemento. Para referenciar um elemento com um ID específico no CSS, você usa uma hashtag (#) seguida do nome do ID:

```
<div id="cabecalho">Este é o cabeçalho.</div>
```

```
#cabecalho {
  font-size: 24px;
  color: red;
}
```

3. **Atributos de elementos específicos**:
   Alguns atributos HTML são específicos para certos elementos. Por exemplo, o atributo "href" em links ou o atributo "src" em elementos de imagem. Você pode usar seletores de elementos para estilizar esses elementos com base em seus atributos:

```
<a href="https://www.example.com">Visite o exemplo</a>
```

```
a[href="https://www.example.com"] {
  text-decoration: none;
  color: green;
}
```

4. **Atributo "style"**:
   O atributo "style" permite que você aplique estilos diretamente a um elemento HTML, inline. No entanto, é uma prática recomendada separar o CSS do HTML sempre que possível. Quando você usa o atributo "style", os estilos definidos nesse atributo têm precedência sobre regras CSS externas.

```
<p style="color: purple;">Este é um parágrafo roxo.</p>
```

Estilização de textos com CSS

- color: altera a cor da fonte. `blue` `red` `yellow`

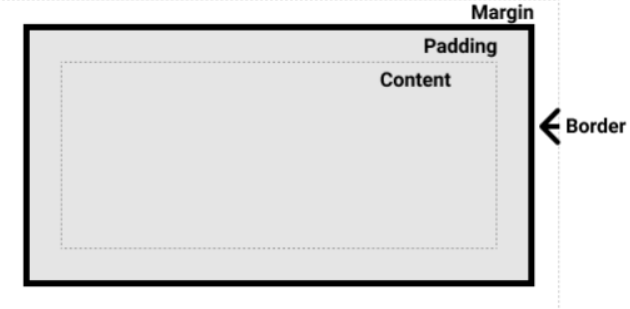
```
p {
  color: red; /* Altera a cor da fonte */
  font-family: Arial; /* Altera a fonte */
}
```

- font-style: ativa o itálico. `italic` `normal` `oblique`
- font-weight: ativa o negrito. `bold` `lighter` `bolder` `normal`

- text-decoration: `none` `underline` `overline` `line-through`

Partes de uma box (caixa)

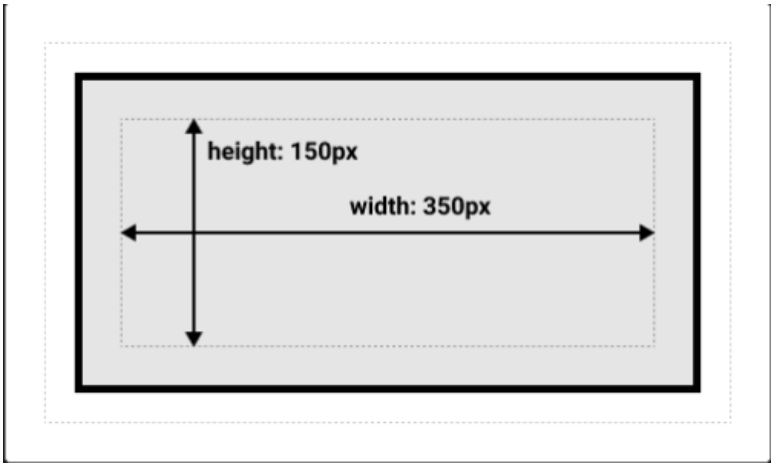
- Content: área do conteúdo, pode ser ajustada pelas propriedades `width` e `height`
- Padding: área em volta do conteúdo, pode ser ajudado pela propriedade `padding`
- Border: envolve o padding e o separa da margem. Pode ser ajustado pela propriedade `border`
- Margin: área que envolve todo o resto. Pode ser ajustada pela propriedade `margin`



Observe o código abaixo

```
.box {
  width: 350px;
  height: 150px;
  margin: 10px;
  padding: 25px;
  border: 5px solid black;
}
```

O espaço efetivo que a caixa vai ocupar de largura vai ser 410px (350+25+25+5+5) e de altura vai ser 210px (150+25+25+5+5)



JavaScript

JavaScript é uma linguagem de programação amplamente utilizada que é principalmente conhecida por sua capacidade de criar interatividade em páginas da web. Ela desempenha um papel fundamental no desenvolvimento web moderno e é suportada pela maioria dos navegadores da web.

JavaScript é uma linguagem de script, o que significa que o código é executado no navegador do cliente. Isto é, quando é feita a requisição HTTP da página WEB, virá também o código JavaScript junto com o HTML, que será interpretado pelo navegador.

Ser **interpretada** significa que não haverá compilação da linguagem. Ela chega como um arquivo de texto ao navegador e é executada imediatamente. É muito parecido com o próprio HTML e CSS.

Sintaxe de JavaScript

1. **Declaração de Variáveis:**

Em JavaScript, você pode declarar variáveis usando as palavras-chave `var`, `let` ou `const`.

```
let idade = 25;      // Declaração de variável usando let
const PI = 3.1415;   // Declaração de constante usando const
```

2. **Estruturas condicionais**

```
if (idade >= 18) {
    console.log("É maior de idade.");
} else {
```

```
    console.log("É menor de idade.");
}
```

3. **Loops**

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

4. **Funções**

```
function saudacao(nome) {
    console.log("Olá, " + nome + "!");
}

saudacao("Maria"); // Chama a função com o argumento "Maria"
```

5. **Objetos**

```
let pessoa = {
    nome: "João",
    idade: 30,
    cidade: "São Paulo"
};

console.log(pessoa.nome); // Acessa uma propriedade do objeto
```

6. **Arrays**

```
let frutas = ["Maçã", "Banana", "Laranja"];
console.log(frutas[0]); // Acessa o primeiro elemento do array (índice 0)
```

Document e events

Em JavaScript, você pode usar o objeto `document` para acessar elementos HTML em sua página da web. A forma mais comum é através do Id do elemento:

```
const meuElemento = document.getElementById('idDoElemento');
```

É possível, agora, alterar propriedades deste meuElemento. Também é possível dizer o que vai acontecer com o elemento em um evento de clique. Por exemplo, no caso abaixo, quando altero o input do html, é chamada a função `updateSliderPWM` que vai printar no console o valor do input.

```
<p><input type="range" onchange="updateSliderPWM()" id="pwmSlider" min="0" max="255"></p>

<script>
function updateSliderPWM(element) {
  const sliderValue = document.getElementById("pwmSlider").value;
  console.log(sliderValue);
}
</script>
```

AJAX

O AJAX permite que você mande requisições a um servidor e recebe uma resposta sem a necessidade de atualizar a página. Para isso, você realiza requisições por meio da classe `XMLHttpRequest()`. O formato básico é este:

```
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML = this.responseText;
    }
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

O método `onload` recebe uma função que executa uma rotina quando a resposta do servidor chegar. O conteúdo da resposta do servidor é acessível por meio de `this.responseText`.

Crie sua requisição pelo método open e utilize a função da forma que desejar:

```
function updateSliderPWM() {
  var sliderValue = document.getElementById("pwmSlider").value;
  document.getElementById("textSliderValue").innerHTML = sliderValue;
  console.log(sliderValue);
  var xhr = new XMLHttpRequest();
  xhr.open("GET", "/slider?value="+sliderValue, true);
```

```
  xhr.send();
}
```

Chamadas assincronas ESP32

A biblioteca utilizada é a ESPAsyncWebServer. Ela traz funcionalidades para criar um servidor que aguarda requisições http. Como ela foi criada para a ESP8266, é necessária a adaptação com a AsyncTCP para a ESP32.

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
```

Instanciamos o server:

```
AsyncWebServer server(80);
```

É aqui que a mágica acontece. Através dos métodos `on` do server, você pode criar os endpoints da aplicação. Os endpoints são os destinos das urls. Você deve criar um endpoint raiz para quem acessar diretamente o endereço IP da ESP32, esta url retornará a página inicial da sua aplicação (`index_html[]`)

```
// Raiz da pagina, retorna o HTML principal, passando pelo processor
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/html", index_html, processor);
});
```

A função `processor` serve para que você possa fazer alterações no arquivo HTML que vai para o cliente. Você pode definir **placeholders** em seu html que receberão valores que você definir nesta função. Placeholders são espécies de variáveis, não fazem parte do html, mas servem de referência para a biblioteca ESPAsyncWebServer. No exemplo abaixo, você estará substituindo o valor do placeholder **SLIDERVALUE** pelo valor da variável sliderValue;

```
String sliderValue = "0";

String processor(const String& var){
  if(var == "SLIDERVALUE"){
    return sliderValue;
  }
```

```
  return String();
}
```

Por fim, só resta criar o endpoint da requisição que está sendo feita via AJAX. A requisição vai vir no seguinte formato (exemplo de um valor 100 enviado):

`/slider?value=100`

A requisição a cima de chama **query**. Isso vai redirecionar a requisição para a função com o endpoint `/slider`. Depois, basta checar se a `value` existe, e capturar o valor dela.

```
server.on("/slider", HTTP_GET, [] (AsyncWebServerRequest *request) {
// Checa se a query possui o parâmetro value
  if (request->hasParam("value")) {
      // Captura o valor da query
      sliderValue = request->getParam("value")->value();
      ledcWrite(0, sliderValue.toInt());
  }
  else {
      sliderValue = "error";
  }
  Serial.print("Valor PWM: ");
  Serial.println(sliderValue);
  request->send(200, "text/plain", "OK");
});

// Inicia o servidor
server.begin();
```

Repare que, para este projeto, não utilizamos o `loop()`. Você poderia utilizar para implementar qualquer rotina que quisesse, mas os endpoints não precisaram ser colocados lá porque a própria biblioteca realiza o trabalho de ficar ouvindo esses endpoint a todo o momento.