Unidade V: Arrays e Strings soulcode.com

**1 Strings**

Strings são úteis para guardar dados que podem ser representados em forma de texto. Podemos usar aspas duplas("), aspas simples(’) ou acento grave(‘) para representar strings.

**1.1 Estruturação de uma string**

Strings são sequências de caracteres isolados. Cada caractere que forma a string está endereçado em uma posição individual e númerica.

Figura 1: Estrutura de uma string



Fonte: os autores.

Os caracteres individuais que formam uma string são indexados, ou seja, possuem um endereço númerico destinado a eles. No caso do Javascript, este endereço é chamado de **index** (índice). A posição propriamente dita é apenas uma forma mais natural de nos referirmos a determinados caracteres.

O elemento na **1º** posição, que está no index **0** é o caractere "J". O elemento na **8º** posição que é o index **7** é o caractere "I", e assim por diante.

**1.2 Comprimento de uma string**

Podemos saber quantos caracteres uma string possui usando a **propriedade** *length*.

Listagem 1: Propriedade length

1 const nome = "Carlos";

2 console.log(nome.length);

3

4 const firstIndex = 0;

5 const lastIndex = nome.length - 1;

Podemos determinar dessa forma o primeiro index (que sempre será 0) e o último index que sempre será o comprimento da string subtraído por 1, pois a contagem de index começa do zero.

**1.3 Acesso a um caractere**

Para acessar um caractere específico da string, podemos usar o esquema de colchetes e passar o índice numérico.

1

Unidade V: Arrays e Strings soulcode.com

Listagem 2: Acesso a caracteres

1 const sobrenome = "Pereira";

2

3 console.log(sobrenome[0]); // "P"

4 console.log(sobrenome[sobrenome.length - 1]); // "a"

5 console.log(sobrenome[1000]); // undefined

**1.4 Percorrendo uma string**

Podemos percorrer uma string utilizando um loop-for. Veja o exemplo a se guir:

Listagem 3: For-loop em strings

1 let meuPet = "Fred";

2

3 for (let i = 0; i < meuPet.length; i++) {

4 console.log(meuPet[i]);

5 }

Este for irá percorrer por todos os index válidos da string, pois está no inter valo *[0, 4[*.

**1.5 Métodos**

Dentro de cada variável String, podemos utilizar diversas funções que nesse contexto recebem o nome de métodos. Elas são capazes de analisar a string, recortar e fazer diversas manipulações úteis.

Listagem 4: Métodos

1 console.log(meuPet.toLowerCase()); // FRED

2 console.log(meuPet.toUpperCase()); // fred

3

4 let arquivo = "batata.mp4";

5 // Verifica se a string termina com a palavra

6 console.log(arquivo.endsWith(".mp4")); // true

7 // Verifica se a string inicia com palavra

8 console.log(arquivo.startsWith("bat")); // true

9

10 let frase = "Eu comi arroz, batata e carne";

11 // Buscar se uma frase inclui um trecho

12 console.log(frase.includes("feijao")); // false

13

14 // Buscar o index do caractere

15 console.log(frase.indexOf("c")); // 3, da esquerda p/ direita 16 console.log(frase.lastIndexOf("c")); // 24, da direita p/ esquerda 17 console.log(frase.indexOf("x")); // -1, nao existe

18 console.log(frase.indexOf("arroz")); // 8, inicio da palavra

19 let novaFrase = frase.replace("carne", "ovo"); // "Eu comi arroz, batata e ovo" 20

21 let palavra2 = "BATATA";

22 console.log(palavra2.slice(1)); // Cortou do 1 ate o final ("ATATA") 23 console.log(palavra2.slice(1, 4)); // Cortar do 1 ate o 3 ("ATA")

2

Unidade V: Arrays e Strings soulcode.com

Existem diversos métodos acerca das strings, resumimos aqui alguns usos práticos dos métodos. Porém você pode conferir todos os métodos disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/ Global\_Objects/String

**1.6 Template literals**

Os template literals são uma sintaxe mais recente introduzida no ECMAScript 6 (ES6), também conhecido como ES2015. Eles permitem a interpolação de strings de forma mais legível e flexível, utilizando a notação de acento grave (backtick, ‘).

Listagem 5: Template literals

1 const nome = 'Joao';

2 const idade = 25;

3

4 const mensagem = `Ola, meu nome eh ${nome} e eu tenho ${idade} anos.`; 5 console.log(mensagem); // Saida: Ola, meu nome eh Joao e eu tenho 25 anos.

Com os template literals, as variáveis e expressões são incorporadas na string usando a sintaxe ${}. Isso torna a interpolação de valores mais simples e legível.

Além disso, os template literals suportam quebras de linha e permitem a inclusão de expressões dentro das interpolações.

Listagem 6: Template literals

1 const preco = 19.99;

2 const quantidade = 3;

3

4 const total = `O total eh:

5 ${preco \* quantidade} reais.`;

6 console.log(total);

7 /\*

8 Saida:

9 O total eh:

10 59.97 reais.

11 \*/

Nesse exemplo, o cálculo ***preco \* quantidade*** é realizado dentro do template literal, e o resultado é interpolado na string resultante.

**2 Arrays**

O objeto Array do JavaScript é um objeto global usado na construção de ’ar rays’: objetos de alto nível semelhantes a listas. Com esta estrutura podemos armazenar uma variedade de dados indexados.

Seu uso é muito prático e importante na programação, pois permite arma zenamento de múltiplos valores.

3

Unidade V: Arrays e Strings soulcode.com

**2.1 Estruturação de um array**

A estruturação de um array é muito similar a das strings, utilizamos index para referenciar cada elemento presente em um array. Para criar um array usamos a sintaxe de colchetes e cada valor é separado por vírgula.

Listagem 7: Declaração de array

1 let figuras = ["coracao", "estrela", "ok", "nao", "joinha", "sorriso"]; 2 console.log(figuras); // representacao em texto

Acima temos um array em que todos os elementos são do tipo String. O con sole.log irá exibir no console do navegador uma representação muito similar à que foi escrita em código.

**2.2 Acessando elementos**

Da mesma forma que uma string, podemos acessar o comprimento de um array: array.length. Para acessar um elemento individual presente no array utilizamos a mesma forma de acesso das strings com colchetes.

Listagem 8: Acesso de elementos

1 console.log(figuras[5]); // "sorriso"

2 console.log(figuras[1]); // "estrela"

3 console.log(numeros[2]); // "ok"

4 console.log(numeros[0]); // "coracao"

5 console.log(numeros[3]); // "nao"

6 console.log(numeros[numeros.length - 1]); // "sorriso"

7 console.log(numeros[1000]); // undefined

**2.3 Alterando o valor de um elemento**

Podemos usar esta sintaxe de acesso por index para alterar o valor presente na posição determinada ou fazer alguma operação importante.

Listagem 9: Alterando elementos

1 let numeros = [3, 1, 5, 7, 2];

2 console.log(numeros);

3

4 numeros[2] = 700; // Alterar o valor armazenado no indice 2, o 5 original se perde

5 console.log(numeros);

6

7 numeros[2]++; // Alterar o valor armazenado, incrementando o que estiver 8 console.log(numeros);

9

10 figuras[5] = "batata";

11 console.log(figuras);

4

Unidade V: Arrays e Strings soulcode.com

**2.4 Percorrendo elementos em sequência**

**2.4.1 For-loop**

Para percorrer um array devemos partir do primeiro índice até o último índice (tamanho total - 1). Podemos fazer um for-loop similar ao da seção 1.4.

Listagem 10: For-loop

1 for (let i = 0; i < numeros.length; i++) {

2 console.log(numeros[i]);

3 }

4

5 for (let i = 0; i < figuras.length; i++) {

6 console.log(`O valor do index ${i} eh ${figuras[i]}`);

7 }

Nos dois exemplos, cada elemento do array será mostrado em sequência. Podemos utilizar este comportamento para fazer diversas manipulações.

**2.4.2 For-of**

O Javascript disponibiliza uma forma do for que é própria para elementos sequenciais como strings e arrays.

Listagem 11: For-of

1 for (let nome of nomes) {

2 // A variavel "nome" assume um elemento diferente a cada laco 3 console.log(nome);

4 }

5

6 // Percorre o array ate o final dele

7 for (let notaAluno of notas) {

8 // Para cada nota no array faca isso

9 console.log(notaAluno);

10 }

**2.5 Métodos**

**2.5.1 Inserir e remover elementos**

Podemos inserir elementos no final de um array e inserir elementos no final utilizando os métodos **pop** e **push**.

Listagem 12: pop x push

1 let listaProdutos = [];

2

3 listaProdutos.push("Smart Band"); // adicionar o elemento no final 4 listaProdutos.push("Bolo de cenoura");

5 listaProdutos.push("Batata remasterizada");

6 // adiciona mais de um por vez

7 listaProdutos.push("Smartphone 2MB", "HD 500kb", "RAM 200B");

8 console.log(listaProdutos);

5

Unidade V: Arrays e Strings soulcode.com

9 let produto1 = listaProdutos.pop(); // remove o ultimo elemento 10 let produto2 = listaProdutos.pop(); // remove o novo ultimo elemento 11 console.log(listaProdutos);

**2.5.2 Checar existência de elementos**

Podemos checar a existência de um elemento no array usando alguns méto dos como **includes** e **indexOf**.

Listagem 13: includes e indexOf

1 console.log(listaProdutos.includes("Bolo de cenoura")); // true 2 console.log(listaProdutos.indexOf("Batata remasterizada")) // 2 3 console.log(listaProdutos.indexOf("Inexistente")) // -1

**2.5.3 Percorrer elementos**

Existe uma forma diferente de percorrer elementos além do for-loop e for-of descritos nas seções anteriores.

Listagem 14: forEach

1 let numeros = [1, 2, 3];

2

3 function mostrarValor(valor) {

4 console.log(valor);

5 }

6

7 // Para cada elemento no array chamamos a funcao mostrarValor passando o elemento 8 numeros.forEach(mostrarValor);

9 // ou

10 numeros.forEach((elemento) => {

11 console.log(elemento);

12 });

Aqui vemos um uso interessante de arrow functions na linha 10, usando a arrow não somos obrigados a definir uma função como acontece nas linhas anteriores, basta apenas determinar o que será feito com cada elemento do array.

**2.5.4 Mapear elementos**

Podemos transformar um array e gerar um novo array com as alterações rea lizadas, é muito útil para diversas manipulações.

Listagem 15: map

1 let nomes = ["Pedro", "Paulo", "Joao", "Lucas"];

2

3 let nomesCaixaBaixa = nomes.map((nome) => {

4 return nome.toLowerCase();

5 });

6

6

Unidade V: Arrays e Strings soulcode.com

7 let nomesCaixaAlta = nomes.map((nome) => {

8 return nome.toUpperCase();

9 });

10

11 let nomesExclamacao = nomes.map((nome) => {

12 return nome + "!";

13 });

Podemos imaginar que a função map do Javascript irá gerar uma entrada transformada, e um novo array será formado de acordo com o que a arrow function retornar.

**2.5.5 Filtrar elementos**

Outra função muito útil é o **filter**, com ela podemos determinar quais elemen tos fazem parte do novo array com base em uma condição. Essa condição deve ser retornada pela arrow interna.

Listagem 16: filter

1 let temps = [29, 5, 10, -4, 15, -20]; // Celsius

2 let tempsPositivas = temps.filter((temp) => {

3 // aplicar esta funcao para item (testar)

4 if (temp > 0) {

5 return true; // t pode fazer parte do novo array

6 } else {

7 return false; // t nao fara parte do novo array

8 }

9 });

A arrow deve retornar um valor boolean, observe que utilizamos um if para retornar true ou false. Porém, poderíamos simplificar a expressão da seguinte maneira:

Listagem 17: filter

1 let temps = [29, 5, 10, -4, 15, -20];

2 let tempsPositivas = temps.filter((temp) => temp > 0);

A expressão temp > 0 já é um valor booleano, basta retornar ele e o filtro será realizado, para os valores true, o elemento fará parte do novo array.

**2.5.6 Outros métodos**

Caso queira saber mais métodos sobre Array recomendo consultar a docu mentação de Arrays do MDN Web Docs.

7