Técnico em Informática para Internet

Unidade Curricular 3: Codificar front-end de aplicações web

Olá! Sejam bem vindos

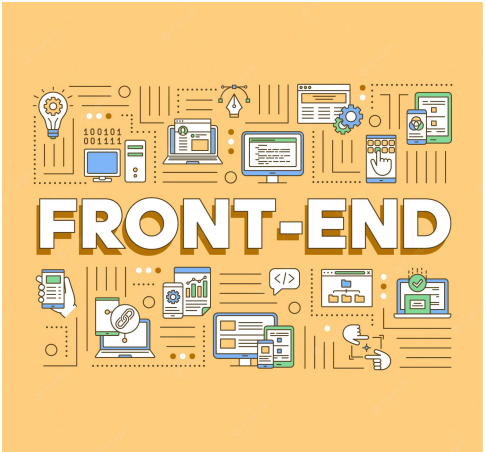
DISCIPLINA/UNIDADE CURRICULAR	UC03 - Codificar front-end de aplicações web 20h
TEMA DA AULA	Linguagens de Programação de Script e Blocos de Back-end
INDICADOR	1. Cria blocos de back-end, utilizando linguagens de programação de script, de acordo com os requisitos do projeto de software
CRONOGRAMA	-Sites estáticos e dinâmicos: conceitos, diferenças, linguagens de script, tecnologias e aplicações. -Programação: conceito, análise de requisitos do projeto e a relação com a codificação em front-end. -Linguagem de Scripts: Sintaxe - operadores, palavras reservadas, identificadores, delimitadores e comentários; variáveis e tipos de dados, estruturas de controle condicional e laços de repetição
ATIVIDADE	Descrição de um exercício prático que os alunos podem fazer para aplicar o que aprenderam

Objetivos da Aula

- Sites estáticos e dinâmicos: conceitos, diferenças, linguagens de script, tecnologias e aplicações.
- Programação: conceito, análise de requisitos do projeto e a relação com a codificação em front-end.
- Linguagem de Scripts: Sintaxe - operadores, palavras reservadas, identificadores, delimitadores e comentários; variáveis e tipos de dados, estruturas de controle condicional e laços de repetição

Programação FrontEnd

- O Front-end está muito relacionado com a interface gráfica do projeto.
- É onde se desenvolve a aplicação com a qual o usuário irá interagir diretamente, seja em softwares, sites, aplicativos, etc.
- Portanto, é essencial que o desenvolvedor tenha uma preocupação com a experiência do usuário.



Sites Estáticos e Dinâmicos: Conceitos, Diferenças, Linguagens de Script, Tecnologias e Aplicações

Introdução

- Os websites podem ser categorizados de muitas maneiras, mas uma das classificações mais comuns é se são estáticos ou dinâmicos.
- Ambos têm suas vantagens e desvantagens e são adequados para diferentes tipos de projetos.
- Um site é uma coleção de páginas organizadas e localizadas em um servidor. Ele pode ser estático ou dinâmico.

Sites Estáticos

- Um site estático é um site que é entregue ao usuário exatamente como armazenado.
- Em outras palavras, a informação que a página contém não muda, a menos que seja alterada manualmente pelo webmaster.
- Eles são geralmente escritos puramente em HTML e CSS.

Aplicações de Sites Estáticos

- Os sites estáticos são comumente usados para páginas que não requerem interação do usuário ou atualizações frequentes, como blogs, portfólios ou sites institucionais.

Sites Dinâmicos

- Um site dinâmico é um site que contém páginas que são geradas em tempo real.
- Esses sites utilizam linguagens de script como JavaScript no front-end, e linguagens como Python, Ruby ou JavaScript (Node.js) no back-end.
- Eles geralmente interagem com um banco de dados.

Aplicações de Sites Dinâmicos

- Os sites dinâmicos são usados quando o conteúdo precisa mudar frequentemente, como em lojas online, redes sociais ou notícias e blogs que são atualizados regularmente.

Comparação entre Sites Estáticos e Dinâmicos

- Em comparação, os sites estáticos são mais rápidos e mais seguros, mas são menos interativos.
- Já os sites dinâmicos são mais interativos, mas requerem mais recursos do servidor.

Programação: Conceito, Análise de Requisitos e a Relação com a Codificação em Front-end

Conceito de Programação

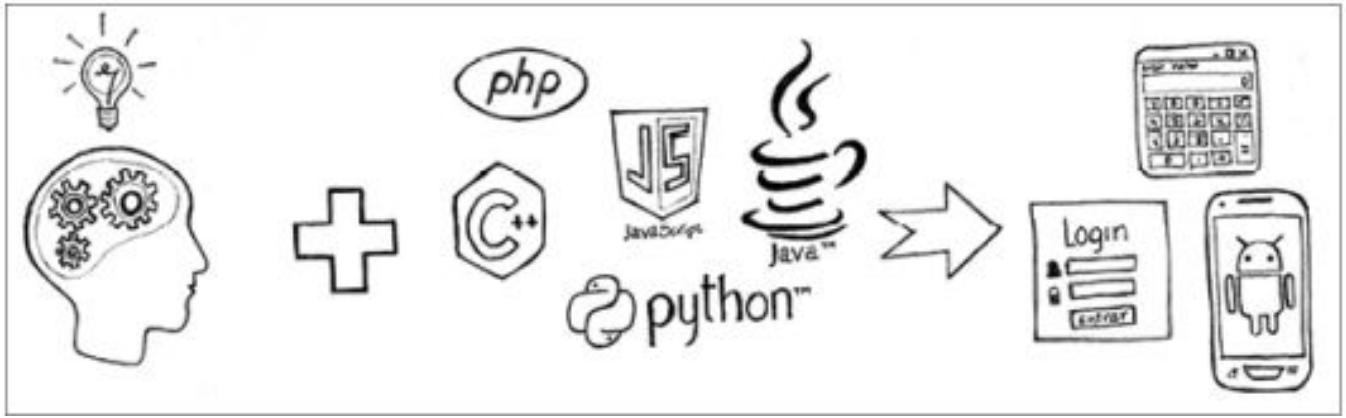
- A programação é o processo de escrever, testar, depurar e manter o código-fonte de programas computacionais.
- O código-fonte é escrito em uma linguagem de programação, onde o programador usa abstrações fornecidas pela linguagem para representar conceitos e operações do domínio do problema.

Conceito de Programação

- Neste universo, você será capaz de construir coisas incríveis.
- Ajudar a melhorar o mundo.
- Desenvolver um sistema para uma empresa aperfeiçoar o gerenciamento de seu negócio.
- Criar um aplicativo a fim de auxiliar pessoas a superarem suas dificuldades
- Seja qual for o seu objetivo, você precisa começar pelo estudo de Algoritmos e Lógica de Programação.

Conceito de Programação

- Se você entender corretamente como criar programas em JavaScript, será muito mais fácil estudar outra linguagem, pois as estruturas e os conceitos básicos são os mesmos.



Algoritmos

- Ao estruturar os passos de um programa, estamos montando um algoritmo.
- Um algoritmo é uma sequência de passos (comandos) a serem executados para a realização de uma tarefa, em um tempo finito.

Lógica de Programação

- Organizar o pensamento e colocar as coisas em ordem são tarefas de lógica de que necessitamos para resolver problemas com o uso do computador.

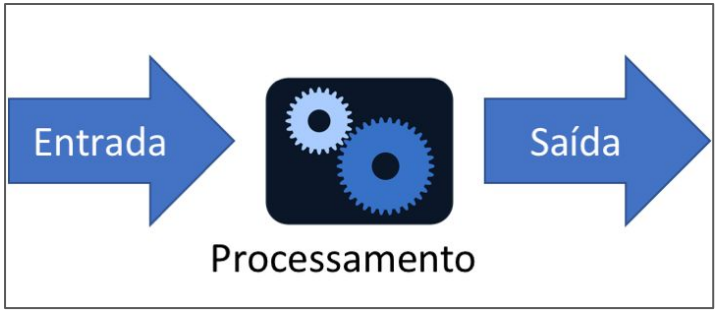
Programa

- Na montagem de um programa, utilizamos comandos sequenciais, comandos para definição de condições e comandos para criação de estruturas de repetição.

Como funciona um computador?

Como funciona um computador?

Exemplo: Funcionamento de um caixa eletrônico



Análise de Requisitos

- A análise de requisitos é a primeira etapa no processo de desenvolvimento de software.
- Ela envolve a determinação das necessidades ou condições a serem atendidas para um novo ou alterado produto.
- Os requisitos devem ser quantificáveis, detalhados, e claros para garantir um produto de sucesso.

A relação da Programação com a Codificação em Front-end

- A programação em front-end envolve a criação de código que é executado no navegador do usuário.
- Isso inclui a exibição de dados na interface do usuário, a manipulação de interações do usuário e a comunicação com o back-end.
- A programação em front-end é responsável por tudo o que o usuário vê e interage em um site.

Importância da Análise de Requisitos no Front-end

- A análise de requisitos é vital na programação de front-end porque ajuda a definir o que a interface do usuário deve fazer e como ela deve se comportar.
- A análise eficaz dos requisitos pode ajudar a criar um produto que atenda às necessidades do usuário e proporciona uma experiência de usuário positiva.

JavaScript

- A linguagem JavaScript foi criada pela Netscape Communications
- Corporation junto com a Sun Microsystems.
- Sua primeira versão foi lançada em 1995.
- Em 1996, a Netscape decidiu entregar o JavaScript para a ECMA (European Computer Manufacturers Association).
- Em 1997, foi lançada a primeira edição da linguagem gerenciada por essa associação.
- A linguagem JavaScript também é chamada de ECMAScript

JavaScript

- JavaScript possui um importante papel no processo de desenvolvimento de páginas para internet, junto com HTML (HyperText Markup Language) e CSS (Cascading Style Sheets).
- No processo de programação de sistemas web, há linguagens que rodam no lado cliente e que rodam no lado do servidor. JavaScript pode ser utilizada em ambos os lados.

Linguagem de Scripts: Sintaxe, Operadores, Palavras Reservadas, Identificadores, Delimitadores e Comentários

Exemplo JavaScript (dentro do HTML)

- Criar um novo projeto
- Criar um arquivo chamado index.html
- O código a ser utilizado será:

```
<script>
  alert("Bem-Vindo ao Mundo JavaScript!");
</script>
```

Exemplo JavaScript (arquivo externo)

- Criar um novo projeto
- Criar um arquivo chamado index.html
- Criar um arquivo chamado script.js
- O código a ser utilizado será:

Sintaxe e Operadores

- A sintaxe é o conjunto de regras que define como um programa em uma linguagem de programação é escrito e interpretado.
- Os operadores são símbolos especiais que realizam operações específicas como adição, subtração, comparação e lógica.

Sintaxe e Operadores - Exemplo

```
// Adição
let soma = 5 + 3;

// Subtração
let subtracao = 5 - 3;

// Multiplicação
let multiplicacao = 5 * 3;

// Divisão
let divisao = 5 / 3;

// Comparação
let isEqual = 5 == 3; // false
```

Palavras Reservadas e Identificadores

- As palavras reservadas são palavras que têm um significado especial na linguagem de programação e não podem ser usadas para outros propósitos.
- Identificadores são os nomes que os programadores dão a elementos como variáveis e funções.

Palavras Reservadas e Identificadores - Exemplo

```
// Palavras reservadas: let,
function
let minhaVariavel = 10;

// Identificador: minhaFuncao
function minhaFuncao() {
  return minhaVariavel;
}
```

Delimitadores e Comentários

- Delimitadores são símbolos que separam diferentes partes do código.
- Comentários são notas deixadas pelos programadores para explicar o que o código está fazendo e não são executados como parte do programa.
- Os comentários não afetam a execução do programa, sendo ignorados pela linguagem.

```
// para comentários de uma linha

/*
  para comentários de várias linhas
*/
```

Delimitadores e Comentários - Exemplo

```
let minhaVariavel = 10; // Delimitador: ;

/* Comentário: Este é um comentário
que ocupa várias linhas */
```

Variáveis e Tipos de Dados

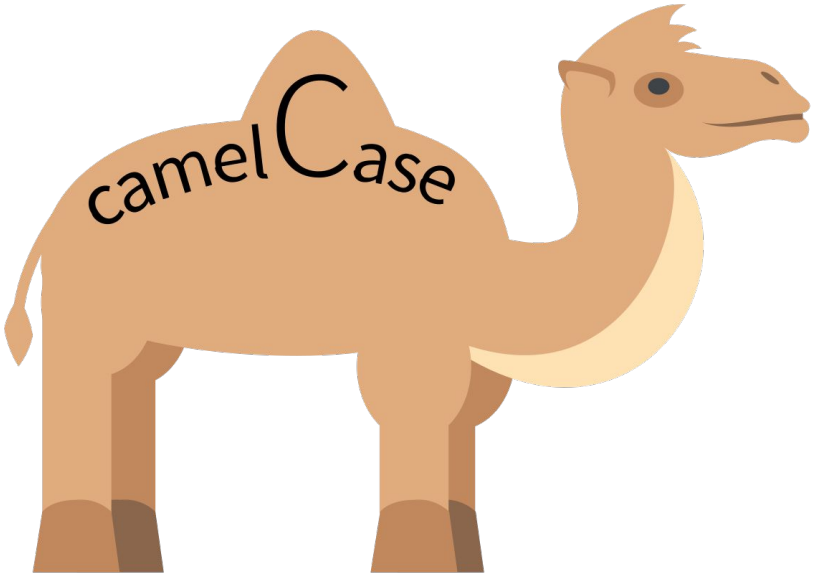
- Variáveis são espaços na memória que guardam um valor.
- Os tipos de dados determinam a natureza do valor que uma variável pode armazenar, como números, strings, valores booleanos ou objetos.
- São exemplos de variáveis manipuladas em um programa: a descrição, a quantidade e o preço de um produto ou, então, o nome, o salário e a altura de uma pessoa.

Variáveis e Tipos de Dados

- Existem regras para criar variáveis, os nomes de variáveis não podem:
  - Conter espaços.
  - Começar por número.
  - Conter caracteres especiais, como +,-,*, /, %, (,),{,},!,@,#.
  - Utilizar nomes de palavras reservadas da linguagem, como function, var, new, for ou return.
  - Variáveis escritas com letras maiúsculas são diferentes de variáveis escritas com letras minúsculas.
  - O uso do caractere “_” é válido.
  - padrão denominado camelcase

Variáveis

- São exemplos de nomes válidos de variáveis:
  - cidade,
  - nota1,
  - primeroCliente,
  - novoSalario,
  - precoFinal,
  - dataVenda.



- Exemplo:
  - let idade = 18;

Constantes

- Para declarar uma constante em um programa, utilize a palavra reservada const seguida pelo nome da constante e da atribuição de valor a ela. É padrão utilizar letras maiúsculas para nomear as constantes.

- Exemplo:
  - const CARTEIRAS = 40;

Variáveis e Tipos de Dados - Exemplo

```
// String
let nome = "John";

// Número
let idade = 30;

// Boolean
let isAdult = true;

// Objeto
let pessoa = {nome: "John", idade: 30};
```

```
<script>
  var fruta = "Banana";
  var preco = 3.50;
  var levar = true;
  var novoValor;
  alert(fruta + " " + preco + " " + levar + " " + novoValor);
</script>
```

Tipos de dados e conversões de tipos

```
<script>
 /*
  Operações envolvendo strings e números
 */
 var a = "20";
 var b = a * 2; // b = 40;
 var c = a / 2; // c = 10;
 var d = a - 2; // d = 18
 var e = a + 2; // e = 202 ???
 alert("e: " + e); // exibe o valor de uma variável
</script>
```

```
<script>
 /*
  Operações envolvendo strings e números
 */
 var a = "20";
 var b = a * 2; // b = 40
 var c = a / 2; // c = 10
 var d = a - 2; // d = 18
 var e = a + 2; // e = 202
 var f = Number(a);
 var g = f + 2; // g = 22
 alert("g: " + g); // exibe o valor de uma variável
</script>
```

Entrada de dados com prompt()

- Para receber dados do usuário, uma das formas possíveis em JavaScript é utilizar o comando(método) prompt(), que exibe uma caixa com um texto e um espaço para digitação.

```
<meta charset="UTF-8">
<script>
 var nome = prompt("Qual é o seu nome?");
 alert("Olá " + nome);
</script>
```

Exercício 01

a) Elaborar um programa que leia um número. Calcule e informe o dobro desse número.

- Entrada de dados: ler um número
- Processamento: calcular o dobro
- Saída: informar o dobro

Exercício 01 - Solução

```
<meta charset="utf-8">
<script>
  // lê um dado de entrada
  var num = prompt("Número: ");
  // calcula o dobro
  var dobro = num * 2;
  // exibe a resposta
  alert("Dobro é: " + dobro);
</script>
```

```
<meta charset="utf-8">
<script>
  // lê um dado de entrada
  var num = Number(prompt("Número: "));
  // calcula o dobro
  var dobro = num + num;
  // exibe a resposta
  alert("Dobro é: " + dobro);
</script>
```

Exercício 02

b) Elaborar um programa que leia dois números. Calcule e informe a soma desses números.

- Entrada de dados: ler dois números
- Processamento: calcular a soma
- Saída: informar a soma

Exercício 02 - Solução

```
<meta charset="utf-8">
<script>
 // lê os números
 var num1 = Number(prompt("1° Número: "));
 var num2 = Number(prompt("2° Número: "));
 // calcula a soma
 var soma = num1 + num2;
 // exibe o resultado
 alert("Soma é: " + soma);
</script>
```

Exercício 03

c) Elaborar um programa que leia o valor de um jantar. Calcule e informe o valor da taxa do garçom (10%) e o valor total a ser pago.

O exemplo de dados do programa serve apenas para ilustrar e reforçar o que é solicitado na descrição do exercício.

```
Valor do Jantar R$: 80.00
Taxa do Garçom R$: 8.00
Total a Pagar R$: 88.00
```

Exercício 03 - Solução

```
<meta charset="utf-8">
<script>
 // lê o valor do jantar
 var jantar = Number(prompt("Valor do Jantar R$: "));
 // calcula os dados
 var garcom = jantar * 0.10;
 var total = jantar + garcom;
 // apresenta as respostas
 alert("Taxa Garçom R$: " + garcom.toFixed(2) + "\nTotal R$: " +
total.toFixed(2));
</script>
```

Exercício 04

d) Elaborar um programa que leia a duração de uma viagem em dias e horas. Calcule e informe a duração total da viagem em número de horas.

Exemplo de dados de entrada e saída do programa (para uma viagem que dura 2 dias + 5 horas).

```
Nº Dias: 2
Nº Horas: 5
Total de Horas: 53
```

Exercício 04 - Solução

```
<meta charset="utf-8">
<script>
 // lê os dados de entrada
 var dias = Number(prompt("Nº Dias: "));
 var horas = Number(prompt("Nº Horas: "));
 // calcula a duração
 var total = (dias * 24) + horas;
 // exibe o total
 alert("Total de Horas: " + total);
</script>
```

Atividade 01

a) Elaborar um programa que leia um número. Calcule e informe os seus vizinhos, ou seja, o número anterior e posterior.

Exemplo:

```
Número: 15
Vizinhos: 14 e 16
```

Atividade 02

b) Elaborar um programa para uma pizzaria, o qual leia o valor total de uma conta e quantos clientes vão pagá-la. Calcule e informe o valor a ser pago por cliente.

Exemplo:

```
Valor da Conta R$: 90.00
Número de Clientes: 3
Valor por cliente R$: 30.00
```

Atividade 03

c) Elaborar um programa para uma loja, o qual leia o preço de um produto e informe as opções de pagamento da loja. Calcule e informe o valor para pagamento à vista com 10% de desconto e o valor em 3x.

Exemplo:

```
Preço R$: 60.00
À Vista R$: 54.00
Ou 3x de R$: 20.00
```

Atividade 04

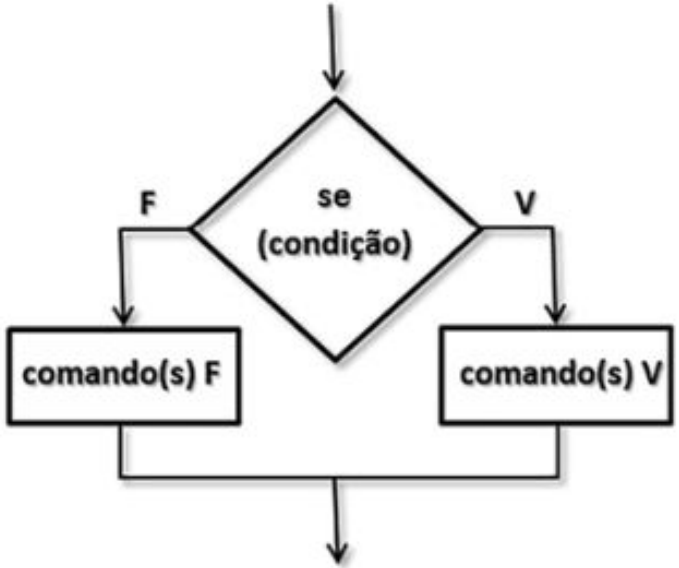
d) Elaborar um programa que leia 2 notas de um aluno em uma disciplina. Calcule e informe a média das notas.

Exemplo:

```
1ª Nota: 7.0
2ª Nota: 8.0
Média: 7.5
```

Estruturas de Controle Condicional

- Estruturas de controle condicional, como if, else e switch, permitem que o código execute diferentes ações com base em diferentes condições.



Estruturas de Controle Condicional - Exemplo

```
// define uma condição simples
if (condição) {
  comandos;
}

// define uma condição de if... else
if (condição) {
  comandos V;
} else {
  comandos F;
}
```

```
// define múltiplas condições
if (condição 1) {
  comandos 1;
} else if (condição 2) {
  comandos 2;
} else {
  comandos 3;
}
```

Estruturas de Controle Condicional - Exemplo

- Testar o valor de um número:

```
let numero = 10;

if (numero > 10) {
  console.log("O número é maior que 10.");
} else if (numero < 10) {
  console.log("O número é menor que 10.");
} else {
  console.log("O número é igual a 10.");
}
```

Estruturas de Controle Condicional - Switch - Case

- As linguagens de programação dispõem de outra estrutura que permite criar condições.
- Trata-se do comando switch... case.
- Ele é útil quando tivermos várias alternativas de

Switch - Case (Exemplo)

```
<script>
	var bairro = prompt("Bairro de Entrega: ");
	var taxaEntrega;
	switch (bairro) {
		case "Centro": taxaEntrega = 5.00; break;
		case "Fragata":
		case "Três Vendas": taxaEntrega = 7.00; break;
		case "Laranjal": taxaEntrega = 10.00; break;
		default: taxaEntrega = 8.00;
	}
	alert("Taxa R$: " + taxaEntrega.toFixed(2));
</script>
```

Operadores Relacionais

Símbolo	Significado
==	Igual. Retorna verdadeiro caso os dados contenham o mesmo conteúdo.
!=	Diferente. Retorna verdadeiro caso os dados contenham conteúdos diferentes.
>	Maior. Pode ser utilizado para comparar números ou palavras. Na comparação de palavras, a classificação alfabética é avaliada.
<	Menor. Também podem ser realizadas comparações de números ou palavras.
>=	Maior ou igual. Os símbolos devem estar nesta ordem (>=)
<=	Menor ou igual. Tenha cuidado com a ordem dos símbolos (<=)

Existem ainda os símbolos de === (estritamente igual) e !== (estritamente diferente). Eles comparam também o tipo do dado em análise. Assim, '5' === 5 retorna falso; e '5' !== 5 retorna verdadeiro.

Operadores Relacionais - Exercício

- Faça um programa que lê duas notas de um aluno e mostre a média.
- Caso a média for maior ou igual a 7 apresentar a mensagem “Parabéns você passou”, se for menor que 7 e maior ou igual a 5 apresentar a mensagem “Estude Mais, você está em recuperação”, e abaixo de 5 mostrar “Você está reprovado!”

Operadores lógicos

Símbolo	Significado
!	Not. Indica negação. Inverte o resultado de uma comparação.
&&	And. Indica conjunção. Retorna verdadeiro quando todas as comparações forem verdadeiras.
||	Or. Indica disjunção. Retorna verdadeiro se, no mínimo, uma das condições definidas for verdadeira.

Operadores lógicos - Exercício

- Faça um programa que solicite o nome, sexo (M - F) e altura.
- Faça o cálculo do peso ideal

Laços de Repetição

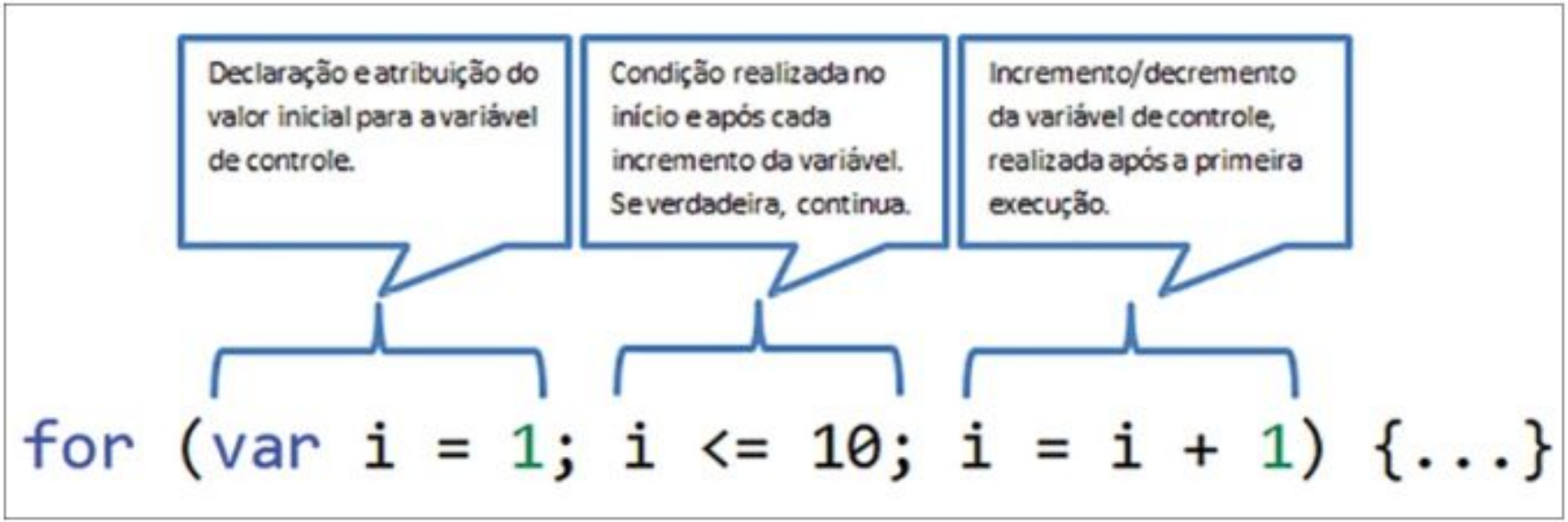
- As estruturas de repetição permitem fazer com que um ou mais comandos em um programa sejam executados várias vezes.
- Essas estruturas, também denominadas laços de repetição ou loops, complementam a programação sequencial e a programação condicional, vistas anteriormente.

Laços de Repetição

- Laços de repetição, como for, while, e do-while, são usados para repetir um bloco de código várias vezes.

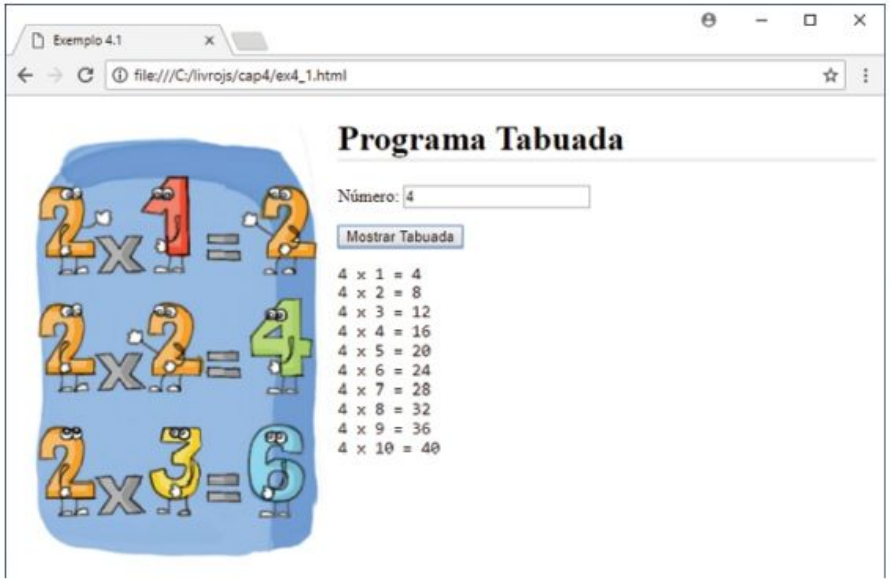
Laços de Repetição

- For



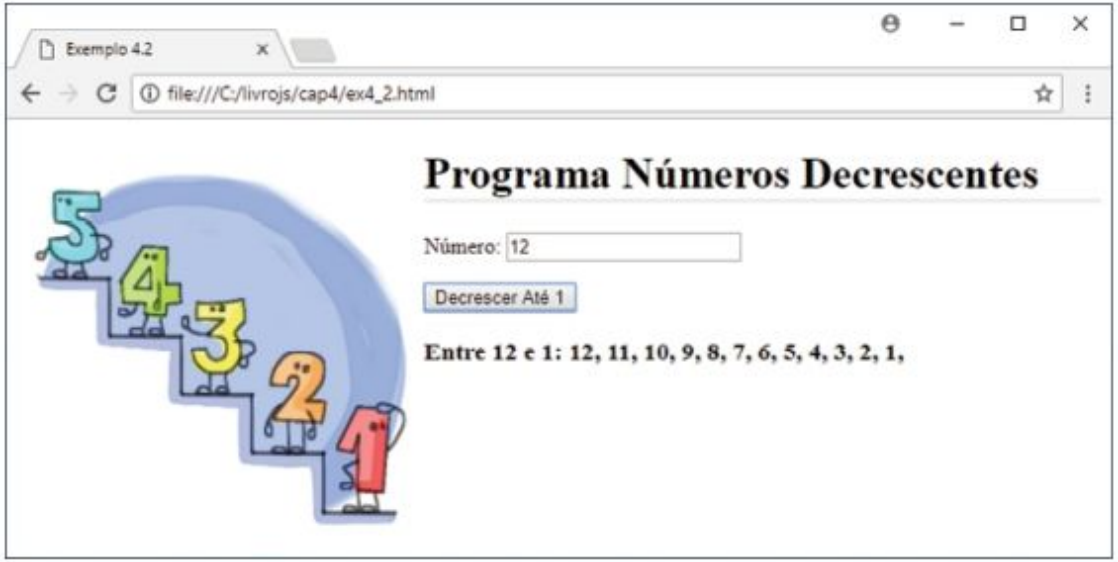
Laços de Repetição

- For (Exemplo 1)
  - Ler um número e apresentar a tabuada desse número.



Laços de Repetição

- For (Exemplo 2)
  - Exemplo de uma montagem de uma estrutura de repetição decrescente, com o valor inicial informado pelo usuário.



Laços de Repetição

- While
  - Um laço de repetição também pode ser criado com o comando while, que realiza um teste condicional logo no seu início, para verificar se os comandos do laço serão ou não executados.

```
while (condição) {
  comandos;
}
```

Laços de Repetição

- Do-While
  - Outra forma de criar laços de repetição em um programa é com a utilização do comando do.. while

```
do {
  comandos;
} while (condição);
```

Laços de Repetição - Exemplo

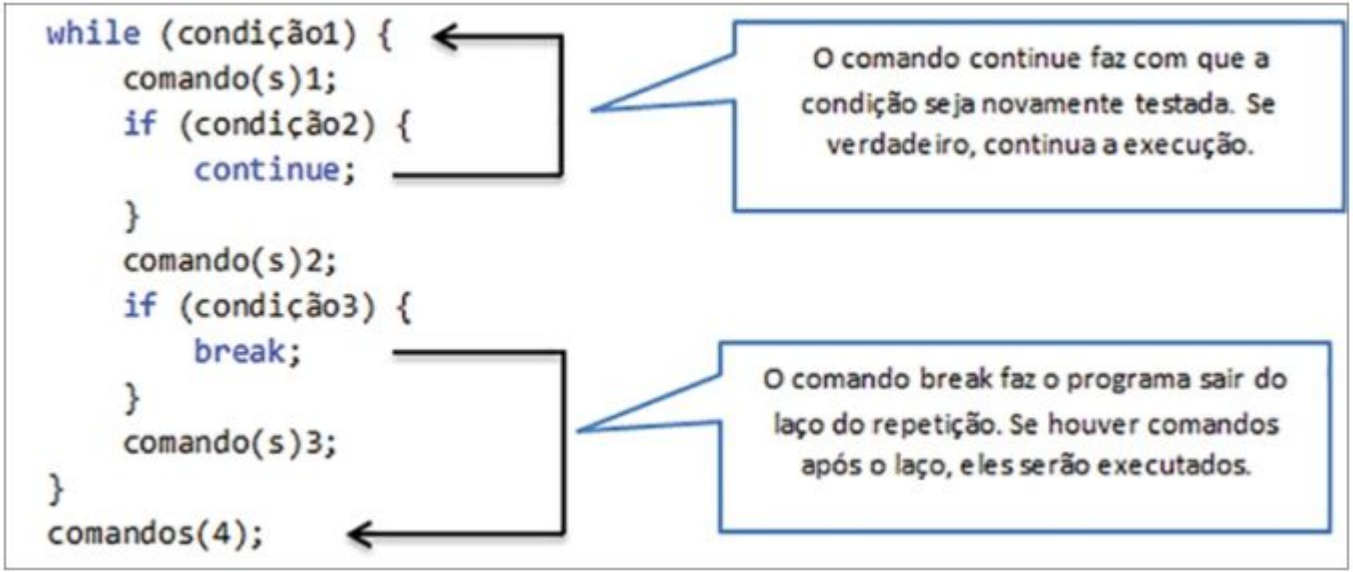
```
// for
for (let i = 0; i < 5; i++) {
    console.log(i);
}

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

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

Interrupções nos laços (break e continue)

- As linguagens de programação dispõem de dois comandos especiais para serem utilizados nas estruturas de repetição. São eles: break e continue.



Contadores e acumuladores

- O uso de contadores e acumuladores em um programa permite a exibição de contagens e totalizações. Essas operações são realizadas sobre os dados manipulados pelo programa.

- Os contadores ou acumuladores possuem duas características principais:
  - A variável contadora ou acumuladora deve receber uma atribuição inicial (geralmente zero).
  - A variável contadora ou acumuladora deve receber ela mesma mais algum valor.

Exercício

- Elaborar um programa que leia o nome de uma fruta e um número. O programa deve repetir a exibição do nome da fruta, de acordo com o número informado. Utilize o “*” para separar os nomes. A Figura ilustra a execução do programa.

Vetores

- Os vetores ou arrays são estruturas que permitem armazenar uma lista de dados na memória principal do computador.
- Eles são úteis para inserir ou remover itens de uma lista de compras ou de alunos de uma turma, por exemplo. Com os vetores é possível recuperar todos os itens inseridos na lista.
- Um índice numérico (que começa em 0) identifica cada elemento da lista. A representação ilustrada na Tabela contém uma lista de itens de um supermercado armazenada no vetor produtos. .

produtos	
0	Arroz
1	Feijão
2	Iogurte
3	Leite
4	Suco
5	Pão

Vetores

- Na linguagem JavaScript, não é necessário indicar o número total de elementos do vetor na sua declaração. Para declarar um vetor em JavaScript, devemos utilizar uma das seguintes formas:

```
var produtos = [];

var produtos = new array();

var produtos = ["Arroz", "Feijão", "Iogurte"];
```

Vetores - Inclusão e exclusão de itens

- Depois de realizarmos a declaração do vetor, podemos gerenciar a lista com a inclusão e a exclusão de itens a esse vetor. Os principais métodos JavaScript que executam essas tarefas estão indicados na Tabela:

`push()`	Adiciona um elemento ao final do vetor.
`unshift()`	Adiciona um elemento ao início do vetor e desloca os elementos existentes uma posição abaixo.
`pop()`	Remove o último elemento do vetor.
`shift()`	Remove o primeiro elemento do vetor e desloca os elementos existentes uma posição acima.

Vetores - Inclusão e exclusão de itens

```
var cidades = ["Pelotas"]; // declara e define conteúdo inicial do vetor
// cidades[0] = "Pelotas"
cidades.push("São Lourenço"); // adiciona cidade ao final do vetor
// cidades[0] = "Pelotas"
// cidades[1] = "São Lourenço";
cidades.unshift("Porto Alegre"); // adiciona ao início e desloca as demais
// cidades[0] = "Porto Alegre"
// cidades[1] = "Pelotas"
// cidades[2] = "São Lourenço";
var retirada = cidades.pop(); // remove a última cidade do vetor
// cidades[0] = "Porto Alegre"
// cidades[1] = "Pelotas"
retirada = cidades.shift(); // remove a primeira e "sobe" as demais
// cidades[0] = "Pelotas"
```

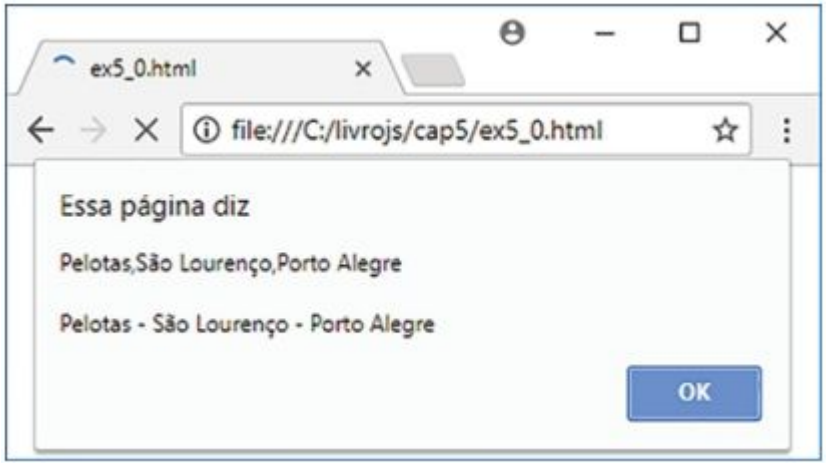
Vetores - Tamanho do vetor e exibição dos itens

```
<script>
 var cidades = ["Pelotas", "São Lourenço", "Porto Alegre"];

 for (var i = 0; i < cidades.length; i++) {
  alert(cidades[i]);
 }
</script>
```

Vetores - toString() e join()

```
<script>
var cidades = ["Pelotas", "São Lourenço", "Porto Alegre"];
alert(cidades.toString() + "\n\n" + cidades.join(" - "));
</script>
```



Vetores - Localizar conteúdo

- Como o número de elementos de um vetor pode ser grande, as linguagens de programação dispõem de alguns métodos para nos auxiliar no controle de seu conteúdo. Um desses controles refere-se à verificação da existência ou não de um conteúdo do vetor. Os métodos indexOf() e lastIndexOf() cumprem esse papel.

```
<script>
  var idades = [5, 6, 8, 3, 6, 9];
  alert(idades.indexOf(6)); // retorna 1
  alert(idades.lastIndexOf(6)); // retorna 4
  alert(idades.indexOf(7)); // retorna -1
</script>
```

Vetores - Vetores de objetos

- Definir um vetor de objetos nos permite realizar operações sobre esse vetor, como classificar os seus elementos por um dos seus atributos.

```
<script>
 var carros = [];
 carros.push({ modelo: "Fusca", preco: 6500 });
 carros.push({ modelo: "Escort", preco: 7800 });

 for (var i = 0; i < carros.length; i++) {
  alert(carros[i].modelo + " - R$: " + carros[i].preco);
 }
</script>
```

Lista de Exercícios

Exercício 01

- Sabendo que o fuso horário da França em relação ao Brasil é de + 5 horas (no horário de verão na França), elaborar um programa que leia a hora no Brasil e informe a hora na França. A Figura ilustra a tela com dados de entrada e saída do programa.

Exercício 02

- Elaborar um programa que leia um número e calcule sua raiz quadrada. Caso a raiz seja exata (quadrados perfeitos), informá-la, caso contrário, informe: ‘Não há raiz exata para ..’. A Figura ilustra uma execução desse programa.

Exercício 03

- Em um determinado momento do dia, apenas notas de 10, 50 e 100 estão disponíveis em um terminal de caixa eletrônico. Elaborar um programa que leia um valor de saque de um cliente, verifique sua validade (ou seja, se pode ser pago com as notas disponíveis) e informe o número mínimo de notas de 100, 50 e 10 necessárias para pagar esse saque.

Exercício 04

- Elaborar um programa que leia um número. Informe se ele é par ou ímpar. A Figura 3.7 ilustra a tela de execução do programa. Para os exercícios, foi utilizada uma figura padrão, mas você pode substituí-la caso tenha interesse.

Exercício 05

- Elaborar um programa que leia a velocidade permitida em uma estrada e a velocidade de um condutor. Se a velocidade for inferior ou igual à permitida, exiba “Sem Multa”. Se a velocidade for de até 20% maior que a permitida, exiba “Multa Leve”. E, se a velocidade for superior a 20% da velocidade permitida, exiba “Multa Grave” – conforme ilustra a Figura.

Exercício 06

- Elaborar um programa para simular um parquímetro, o qual leia o valor de moedas depositado em um terminal de estacionamento rotativo. O programa deve informar o tempo de permanência do veículo no local e o troco (se existir), como no exemplo da Figura 3.9. Se o valor for inferior ao tempo mínimo, exiba a mensagem: “Valor Insuficiente, Considerar os valores/tempos da Tabela (o máximo é 120 min).

Valor R$	Tempo (min)
1,00	30
1,75	60
3,00	120

Exercício 07

- Elaborar um programa que leia três lados e verifique se eles podem ou não formar um triângulo. Para formar um triângulo, um dos lados não pode ser maior que a soma dos outros dois. Caso possam formar um triângulo, exiba também qual o tipo do triângulo: Equilátero (3 lados iguais), Isósceles (2 lados iguais) e Escaleno (3 lados diferentes). A Figura exibe um exemplo de execução do exercício.

Exercício 08

- Digamos que o número de chinchilas de uma fazenda triplica a cada ano, após o primeiro ano. Elaborar um programa que leia o número inicial de chinchilas e anos e informe ano a ano o número médio previsto de chinchilas da fazenda. Validar a entrada para que o número inicial de chinchilas seja maior ou igual a 2 (um casal). A Figura exibe a página com um exemplo de saída do programa.

Exercício 09

- Elaborar um programa que leia um número e verifique se ele é ou não perfeito. Um número dito perfeito é aquele que é igual à soma dos seus divisores inteiros (exceto o próprio número). O programa deve exibir os divisores do número e a soma deles. A Figura exibe a página do programa com um exemplo de número perfeito.

Exercício 10

- Elaborar um programa que leia um número e exiba estrelas na página, em linhas diferentes. A cada nova linha, o número de estrelas deve ser incrementado. Observação: caso você informe um valor alto, as linhas podem ultrapassar o tamanho da imagem e se alinhar à margem esquerda da página. Para evitar que isso ocorra, crie uma nova classe no arquivo estilos.css, que define uma flutuação a direita para essa imagem. A Figura ilustra uma execução do programa desse exercício.

Exercício 11

- Desenvolva uma fábrica de estrelas, onde informe o número de estrelas e apresenta conforme o exemplo abaixo:

Exercício 12

- Desenvolva um programa que verifique se o número informado é primo:

Exercício 13

Exercício 14

- Nosso programa deve controlar a lista de atendimentos dos pacientes de um consultório odontológico – como se fosse um painel em exposição em uma tv do consultório.

Exercício 15

- Jogo Descubra o Número

Exercício 16

- Nosso programa é para a “Revenda Herbie”, que vai armazenar em um vetor de objetos o modelo e o preço dos carros disponíveis na revenda. Um botão de “Filtrar por Preço” vai solicitar o valor máximo de que um determinado cliente dispõe para a compra do veículo.

Referências

ALMEIDA, Flávio. Cangaceiro JavaScript: Uma aventura no sertão da programação. São Paulo: Ed. Casa do Código, 2017.

FREEMAN, Eric; Robson, Elisabeth. Use a Cabeça! Programação Javascript. Rio de Janeiro. Editora Alta Books, 2016.

IEPSEN, Edécio Fernando. Lógica de Programação e Algoritmos com JavaScript: Uma introdução à programação de computadores com exemplos e exercícios para iniciantes. São Paulo: Novatec, 2018.

Bibliografia Complementar

ALVES, William Pereira. Desenvolvimento de aplicações web com Angular. Rio de Janeiro: Ed. Alta Books, 2018.

EIS, Diego. Guia Front-End: O caminho das pedras para ser um devFront-End. São Paulo: Ed. Casa do Código, 2015.

MOLINARI, Willian. Desconstruindo a Web. As tecnologias por trás de uma requisição. São Paulo: Ed. Casa do Código, 2016.

Nieradka, Itamar Pena. Javascript + CSS + DOM: Desenvolvimento para Web. Rio de Janeiro: Ed. Novaterra, 2016.