**Exercício de Fixação JavaScript – Word e GitHub**

Objetivo: Criar um documento organizado para responder os exercícios.

1. Abra o Word e crie um novo documento.

2. Salve como Exercício\_Fixacao\_JS.docx na pasta javascript-fixacao.

3. Estrutura sugerida dentro do Word:

Cabeçalho:

**Exercício de Fixação JavaScript**

**Nome: Anefátima Bezerra da Silva Figueiredo**

**Data: 02/10/2025**

**Seções no documento**:

**● 1. Variáveis e Tipos**

**var:** Tem escopo de função, sofre hoisting e pode ser redeclarada

**let:** Têm escopo de bloco, não sofre hoisting completo e não pode ser redeclarada no mesmo escopo

**const**: Tem escopo de bloco como let, mas não pode ser reatribuída (valor constante)

Qual a diferença entre var, let e const?

● Liste os tipos primitivos do JavaScript com exemplos.

● Qual a diferença entre null e undefined?

**● 2. Operadores e Expressões**

● operadores matemáticos: +, -, \*, /.

● operadores lógicos: &&, ||, !.

**● 3. Estruturas de Controle**

As estruturas if, else if e else são usadas para controle de fluxo em Java, permitindo que diferentes blocos de código sejam executados baseados em condições específicas.

**if:** código executado se a condição for verdadeira.

**else**: código executado se a condição for falsa.

**if e else.** (exemplo)

if (condição1) {

// executado se condição1 for verdadeira

} else if (condição2) {

// executado se condição2 for verdadeira

} else if (condição3) {

// executado se condição3 for verdadeira

} else {

// executado se nenhuma condição for verdadeira

}

**● 4. Loops e Repetições**

* Os tipos de loops: for, while, do...while.

Os loops são estruturas fundamentais que permitem executar um bloco de código múltiplas vezes, automatizando tarefas repetitivas

**for** = Características:

* Uso ideal: Quando sabemos o número exato de iterações
* Controle: Variável de contador com escopo local
* Estrutura: Inicialização, condição e incremento em uma linha

**While** =Características:

* Uso ideal: Quando não sabemos quantas iterações serão necessárias
* Condição: Verificada ANTES da execução do bloco
* Risco: Loop infinito se a condição nunca se tornar falsa

**do-while = Características:**

* Uso ideal: Quando o bloco deve executar PELO MENOS UMA VEZ
* Condição: Verificada DEPOIS da execução do bloco
* Garantia: Execução mínima de uma iteração

**● 5. Funções**

Em Java, funções são chamadas de métodos e são blocos de código que realizam uma tarefa específica e podem ser reutilizados. Elas são fundamentais para criar código organizado, reutilizável e de fácil manutenção em Java.

**Principais Vantagens das Funções:**

Reutilização - Evita código duplicado

Modularidade - Divide problemas complexos em partes menores

Manutenibilidade - Facilita encontrar e corrigir erros

Testabilidade - Permite testar unidades isoladas

Legibilidade - Torna o código mais claro e organizado

**● 6. Mini-casos práticos**

public class ValidadorFormulario {

// ✅ Valida se email tem formato correto

public static boolean validarEmail(String email) {

return email != null &&

email.contains("@") &&

email.contains(".") &&

email.length() >= 5;

}

// ✅ Valida força da senha

public static String validarSenha(String senha) {

if (senha == null || senha.length() < 8) {

return "Senha deve ter pelo menos 8 caracteres";

}

if (!senha.matches(".\*[A-Z].\*")) {

return "Senha deve conter pelo menos uma letra maiúscula";

}

if (!senha.matches(".\*[0-9].\*")) {

return "Senha deve conter pelo menos um número";

}

return "Senha válida";

}

// ✅ Valida idade para cadastro

public static boolean validarIdade(int idade) {

return idade >= 18 && idade <= 120;

}

// ✅ Teste prático

public static void main(String[] args) {

String email = "usuario@email.com";

String senha = "Senha123";

int idade = 25;

System.out.println("Email válido: " + validarEmail(email));

System.out.println("Validação senha: " + validarSenha(senha));

System.out.println("Idade válida: " + validarIdade(idade));

}

}

**● 7. Reflexão**

**● Por que conhecer tipos e operadores ajuda a programar melhor?**

R: Conhecer tipos e operadores é fundamental para programação eficiente

● **Por que usar console.log() é importante para debug?**

R: O console.log() (ou System.out.println() em Java) é uma ferramenta fundamental para debugging, especialmente para desenvolvedores iniciantes. Aqui estão as razões principais:

O console.log() é a ferramenta perfeita para:

* Iniciantes entenderem o fluxo de código
* Debug rápido e simples
* Prototipagem e experimentação
* Ensino e aprendizado

**● Como planejar variáveis, funções e loops antes de programar?**

Planejar antes de codificar é necessário analisar o problema e buscar uma solução.

**1. Análise do Problema - Passo Fundamental**

**Perguntas-Chave:**

* O que o programa precisa fazer?
* Quais dados ele vai processar?
* Como os dados serão transformados?
* Quais são as entradas e saídas?

**Antes de Programar:**

* Entendi completamente o problema?
* Liste todas as entradas necessárias?
* Liste todas as saídas esperadas?
* Identifiquei os dados que preciso armazenar?
* Planejei as variáveis com tipos adequados?
* Dividi o problema em funções menores?
* Identifiquei onde preciso repetir ações?
* Considerei casos especiais e erros?

**Para Cada Função:**

Nome descritivo do que faz?

Parâmetros de entrada definidos?

Tipo de retorno definido?

Exceções tratadas?

**Para Cada Loop:**

* Condição de parada clara?
* Variável de controle definida?
* Incremento/decremento adequado?
* Risco de loop infinito eliminado?

**Exemplo:**

Caso Prático Completo: Sistema de Notas

Passo 1: Análise do Problema

text

PROBLEMA: Calcular média e situação de alunos

ENTRADAS: Nome, 4 notas

SAÍDAS: Média, situação (Aprovado/Reprovado)

Passo 2: Planejar Variáveis

java

// 📊 VARIÁVEIS PLANEJADAS:

public class SistemaNotas {

// Entradas

String nomeAluno;

double nota1, nota2, nota3, nota4;

// Processamento

double media;

double somaNotas;

int quantidadeNotasValidas;

// Saída

String situacao;

boolean aprovado;

// Controle

int totalAlunos;

int aprovados;

int reprovados;

}

Passo 3: Planejar Funções

java

// 🎯 FUNÇÕES PLANEJADAS:

public class CalculadoraNotas {

// ✅ Calcular média de 4 notas

public double calcularMedia(double n1, double n2, double n3, double n4) {

// (n1 + n2 + n3 + n4) / 4

}

// ✅ Determinar situação

public String determinarSituacao(double media) {

// if media >= 7 → "Aprovado" else "Reprovado"

}

// ✅ Validar nota (0-10)

public boolean validarNota(double nota) {

// return nota >= 0 && nota <= 10

}

// ✅ Gerar relatório

public void gerarRelatorio(String nome, double media, String situacao) {

// Imprimir resultados formatados

}

}

Passo 4: Planejar Loops

java

// 🔄 LOOPS PLANEJADOS:

// Loop principal para múltiplos alunos

while (adicionarMaisAlunos) {

// Coletar dados de um aluno

// Calcular média

// Determinar situação

// Exibir resultado

// Perguntar se continua

}

// Loop para validar notas

for (int i = 0; i < 4; i++) {

do {

// Solicitar nota

// Validar nota

} while (!notaValida);

}

// Loop para estatísticas

for (Aluno aluno : listaAlunos) {

if (aluno.isAprovado()) {

aprovados++;

} else {

reprovados++;

}

}

**● Explique == e ===.**

== (igualdade solta): Compara valores convertendo tipos se necessário

=== (igualdade estrita): Compara valores E tipos (não faz conversão)