Exercício de Fixação JavaScript

Nome: Ronaldo Gonçalves da Silva

Data: 06 de outubro de 2025

1 Variáveis e Tipos

1.1 O que são variáveis e tipos?

R = Variáveis: são espaços vazios criados na memória dum sistema para receber algo (valores). Tipos: são a classificações desses valores que preenche esse espaço vazio, ora criado. Exemplo: cria-se num terreno (sistema) um galpão (espaço vazio) para guardar carros (valores).

1.2 Qual a diferença entre var, let e const?

R = Todas são palavras chaves (comando) do JavaScript (JS) que nos permite criar as variáveis. A primeira (var) não se recomenda mais. Noutras palavras desencoraja o seu uso. A segunda (let), como dito, é um comando que permite criar inúmeras variáveis cujos valores podem ser mudados constantemente, que ao contrário da terceira e última (const) não pode. Essa é a principal diferença entre elas. Const: cria inúmeras variáveis que recebem um único valor invariável, noutras palavras o valor é imutável. Já let e var, ambas criam inúmeras variáveis cujos valores são mutáveis.

1.3 Liste os tipos primitivos do JS com exemplos

R = Os tipos primitivos são: string, number, boolean, undefined, null, symbol e bigInt.

Definição/exemplo

1 String: define texto entre aspas / let nome = ‘Ronaldo’;

2 Number: define números inteiros ou decimais / let idade = 43;

3 Boolean: define se é falso(false) ou verdadeiro (true) / 2 + 2 == 5 false ou 2 + 2 == 4 true;

4 Undefined: é um tipo e ao mesmo tempo um valor que o JavaScript automaticamente atribui quando uma variável é declarada, mas não inicializada com um valor específico / let nome;

5 Null: define um valor atribuído a uma variável (espaço vazio) / let idade = null;

6 Symbol: define um identificador único / let id = Symbol(“id”);

7 BigInt: define número grande / 738438766n

1.4 Qual a diferença entre null e undefined?

R = Null espaço vazio proposital (intencional). Undefined é um espaço vazio, mas não proposital. Ambas na ótica do usuário. Já na visão do desenvolvedor, ambas, são espaços vazios, porém preenchidos com valores diferentes e em momentos diferentes. Imaginemos um formulário. Iremos pegar um item comumente opcional contato telefônico. O usuário pode preencher ou não o campo de contato telefônico. Se preenche retorna o valor preenchido, se não, retorna algo que o desenvolvedor programou que pode ser o valor null ou uma mensagem qualquer (por exemplo: não informado). O undefined na ótica do usuário é um espaço vazio não intencional, porque o usuário não tem controle, não decidiu ainda se tal espaço irá ficar vazio. Na ótica do desenvolvedor, este decidiu que iria ficar vazio, mas o valor undefined é preenchido pelo JavaScript (JS), enquanto o null é por ele mesmo, mas em momentos diferentes. Por exemplo: na criação dum formulário inúmeras variáveis são criados (nome, idade, endereço etc.), ou seja, foram criados vários espaços intencionalmente vazios (na ótica do desenvolvedor), mas que o JS atribui o valor undefined automaticamente. Quando o usuário preenche alguns campos outro não, esse campo não preenchido intencionalmente vai receber um valor atribuído pelo desenvolvedor no momento da programação que pode ser null ou uma mensagem 'não informado'.

1.5 Explique == e ===.

R = Primeiro igualdade não estrita, analisa apenas o valor se é igual ou não. Exemplo: 5 == "5", retorna true, por ser verdade que 5 é igual a 5. Segundo igualdade restrita, analisa não só o valor, mas também o tipo, no mesmo exemplo 5 === "5", retorna false, porque embora o valor seja igual, mas os tipos são diferentes um é do tipo Number e outro é do tipo String. O primeiro também é conhecido como igualdade solta (não recomendado usar), ou seja, permite o JS converter coercitivamente alguns valores e/ou tipo.

2 Operadores e Expressões

2.1 Liste operadores matemáticos / exemplo

1 +: adição ou concatenação / 2 + 3 retorna 5;

2 -: subtração / 2 – 3 retorna -1;

3 *: multiplicação / 2 * 3 retorna 6;

4 /: divisão / 7 / 2 retorna 3.5;

5 %: resto da divisão / 7 % 2 retorna 1 (resto);

6 **: exponenciação / 2**3 retorna 8;

7 ++: incremento / adiciona 1 a determinado valor. Ex.: se o valor é 2 o ++ adiciona 1 retornando 3 (2+1);

2.2 Liste operadores lógicos / exemplos

1 &&: e lógico / let eLogico = true && true; retorna true

2 ||: ou lógico, disjunção equivale a ou/ou; let ouLogico = true || false; retorna true

3 !: negação lógica; let naoLogico = !true; retorna false

2.3 Preveja os resultados

1 – “5” + 2; retorna 52

2 – true + 1; retorna 2

3 Estruturas de Controle

3.1 Explique if, else if e else

R = Todos fazem parte duma estrutura de controle em JS. O if quer dizer ‘se’, o else ‘senão’, logo o if else quer dizer ‘senão se’. Todas são expressões que denotam condições. Tipo: se isso, execute assim, senão execute desse jeito (estrutura composta). Se isso, execute assim, senão se isso, execute assim 2, senão execute desse jeito (estrutura aninhada). Se isso, execute assim (estrutura simples).

A estrutura de controle simples é a que segue

```
if (condição) {
        Linha ou bloco código se a condição for atendida
}
```

A estrutura de controle composta é a que segue

```
if (condição) {
        linha ou bloco de código se a condição for atendida
} else {
        linha ou bloco de código se a condição não for atendida
}
```

A estrutura de controle composta e aninhada

```
if (condição) {
        linha ou bloco de código se a condição for atendida
} else if (condição 2) {
        linha ou bloco de código se a primeira condição não for atendida e a condição 2 for atendida
```

} else {

linha ou bloco de código se as duas condições não forem atendidas

}

```
// Ambiente de teste JS

// Sistema de verificação de maior idade

let idade = 64;
if (idade <= 12) {
    console.log("Você é uma criança!");
} else if (idade < 18) {
    console.log("Você é adolescente!");
} else if (idade < 65) {
    console.log("Você é adulto!");
} else {
    console.log("Você é idoso!");
}
```

3.2 Como usar o switch?

R = trata-se duma outra estrutura de controle um tanto quanto diferente do if, else if, else, mas com lógica semelhantes. Sua estrutura (sintaxe) é a que segue:

Switch (geralmente a variável que se vai comparar, mas pode ser uma expressão também que se vai comparar com os parâmetros) {

case parâmetro1:

código se atende o parâmetro1

break;

case parâmetro2:

código se atende o parâmetro2

break;

case parâmetro3:

código se atende o parâmetro3

break;

default:

código se não atendeu a nenhum parâmetro (

break;

}

Veja: dentro do switch foi uma expressão (true) e não a variável idade. Se coloco a expressão idade ao invés do true não funcionaria correto. Ou seja, a sintaxe estaria correta, mas a lógica não.

3.3 Escreva um exemplo de verificação de maior idade

```
// Sistema de verificação de maior idade com switch

let idade = 65;

switch (true) {
    case (idade <= 12):
        console.log("Você é uma criança!");
        break;
    case (idade < 18):
        console.log("Você é adolescente!");
        break;
    case (idade < 65):
        console.log("Você é adulto!");
        break;
    default:
        console.log("Você é idoso!");
        break;
}
```

4 Loops e Repetições

4.1 Liste os tipos de loops

1 for: estrutura de repetição, com teste lógico no início. Sintaxe:

for (variável; condição; incremento) {

linha ou bloco de código;

incrementador;

}

2 while: estrutura de repetição semelhante a anterior, com teste lógico no início. Sintaxe:

while (condição) {

linha ou bloco de código;

incrementador;

}

3 do while: estrutura de repetição semelhante a anterior, com teste lógico no final. Sintaxe:

do {

linha ou bloco de código;

incrementador;

} while (condição);

4.2 Escreva mentalmente como imprimir números de 1 a 5.

```
// Imprimir número de 1 a 5 com for

for (i = 1; i <= 5; i++) {
    console.log(i);
}


// Imprimir número de 1 a 5 com while
/
let i = 0;
while (i <= 5) {
    console.log(i);
    i++;
}
*/

// Imprimir número de 1 a 5 com do while
/
let i = 0;
do {
    console.log(i);
    i++;
} while (i <= 5);
*/
```

4.3 Explique o break e quando usá-lo

R = palavra chave do bloco de código denominado switch. Informa ao JS que para de executar o programa naquele ponto ou saia do programa naquele ponto. Caso não use o break, o bloco de código será plenamente executado.

5 Funções

5.1 O que é uma função?

R = Trata-se dum bloco de código que pode ser usado, reutilizado inúmeras vezes, bastando para tanto chamá-lo. Sua sintaxe é a que segue:

function nomeDaFunção (parâmetro se houver) {

linha ou bloco de código;

return;

}

5.2 Diferença entre função declarada e função expressa.

R = Função declarada tem a sintaxe: function nome(){}. Já a função expressa tem com sintaxe: let nome = function(){}

Outra diferença, a primeira chamamos pelo nome da função. A segunda chama pelo nome da variável que guarda a função. Nos dois casos acima o uso ocorre quando chamamos: nome(); como ambas tem o mesmo nome, ou seja, tanto a variável, como a função tem o mesmo nome, a chamada se dá de forma igual.

Outra diferença: a primeira pode ser chamada antes, a segunda não.

5.3 Crie uma função que recebe um nome e retorna saudação.

R = function cumprimentar (nome) {

return `Olá, ${nome}, prazer te conhecer!`;

}

console.log(cumprimentar(“Gisele”));

```
function cumprimentar (nome) {
    return `Olá, ${nome}, prazer te conhecer!`;
}
console.log(cumprimentar("Gisele"));
```

No dois casos saída esperada: Olá, Gisele, prazer te conhecer!

6 Mini-casos práticos

6.1 Verificação de número par ou ímpar.

```
let numero = 1135;
if (numero % 2 === 0) {
    console.log(`É par!`);
} else {
    console.log(`É ímpar!`);
}
// É ímpar!
```

6.2 Criação mental de uma lista de compras (array)

Carrinho de compras = [‘feijão’, ‘arroz’, ‘macarrão’, ‘farinha’, ‘açúcar’, ‘café’]

6.3 Somar números de 1 a 10 usando loop.

```
let n1 = 1;
let n2 = 10;
let soma = 0;
let listaNumeros = [];

for (let i = n1; i <= n2; i++) {
    soma += i;
    listaNumeros.push(i);
}
console.log( `A soma dos números 👉 [${listaNumeros.join(', ')}] é igual a: ${soma}.`);

A soma dos números 👉 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] é igual a: 55.
```

7 Reflexão

7.1 Por que conhecer tipos e operadores ajuda a programar melhor?

R = Para tratamentos de dados: para tratar algo é interessante conhecê-los bem e saber como se comporta em determinadas situações. Por exemplo: imagine um programa na web que vai solicitar do usuário dois números para soma, subtração etc. Seja via prompt(), seja via input, embora o usuário digite números, estes são strings para o JS. Haverá necessidade de tratar esses dados. Convertê-los para o

tipo Number. Pois, só assim é que poderemos somar os referidos números. Do contrário haveria concatenação, e não adição como seria esperado. Daí a importância de conhecer os tipos e os operadores para sabe-los manipular corretamente.

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

R = Primeiro porque permite identificar onde está mais ou menos o erro do código. Segundo, permite conhecer o caminho que o programa executa. Noutras palavras, a ordem de execução do programa. Terceiro permite testarmos o programa antes de colocá-lo na web etc.

7.3 Como planejar variáveis, funções e loops antes de programar?

R = Escrever pseudocódigo em português; identificar variáveis necessárias e o que precisa ser armazenado nelas; planejar funções (se necessário), que ações o programa precisa fazer etc.; definir loops onde houver repetição de tarefas; listar entradas e saídas, tipo o que recebe e o que retorna.