01 - Alguns conceitos primordiais de Javascript

AVISO: versão em PDF disponível para download está nessa mesma pasta.

AVISO 02: Só olhar em TI nunca é o suficiente, é prudente que o aluno escreva e teste o conteúdo abaixo.

Zero: Para que serve Javascript?

Javascript nasceu e evoluiu principalmente para servir a área de Web, independente de ter ficado famoso e ter se espalhado para outras áreas como Mobile e no desenvolvimento de jogos. Muito do seu trabalho é focado justamente em manipular a aparência de sistemas o qual o **usuário vai ver** (a parte do **front-end** dos sites). Caso não seja isso, é na manipulação de dados no back-end, **o qual não é o foco da disciplina, mas inevitávelmente dará ensinamentos nesse assunto também. Enfim, Javascript é muito útil pessoal.** 😊

Nossa missão aqui será entender tipos de dados e estruturas as quais o JS usa para realizar as tarefas citadas acima. Com objetivo de ajudar o aluno no processo das aulas.

1. Variáveis e constantes: o quê são e como funcionam?

Para aqueles mais novatos em programação, apenas pensem dessa forma: variáveis e constantes são *lugares onde podemos armazenar e depois até modificar determinadas informações importantes*.

1.1 Como criar novas variáveis e constantes:

```
/* Váriável antiga: Leia a DICA abaixo... */
var maiorPalavraPTBR = 'pneumoultramicroscopicossilicovulcanoconiótico'

/* Váriáveis modernas: */
let meuNome = 'Ivan Guimarães dos S. da Silva'
let minhaIdade = 22

/* Constantes: */
const meuNascimento = '25/04/1999'
const meusDadosPessoais = { cpf: '000.000.000-00', rg: '00.000.000-0' }
```

```
/* COMO MOSTRAR ESSAS COISAS NA TELA: */
console.log(meuNome)
console.log(minhaIdade)
console.log(meuNascimento)
console.log(meusDadosPessoais)
```

DICA: Assim como a maior palavra do nosso dicionário, usar VAR é uma desgraça e gera problemas. Usar LET é bem mais seguro e livre de problemas, entendam LET como VAR só que

melhor.

1.2 Que tipo de valores podemos por em variáveis ou constantes:

A variedade é bem grande e útil!

1.2.1 Strings e números

Podemos ter letras ou números:

```
const Primeira_Letra_Do_Meu_Nome = 'I'
let minhaIdade = 22
```

1.2.2 Listas ou Arrays

Podemos criar listas contendo números importantes:

```
const paresMenoresQueVinte = [2,4,6,8,10,12,14,16,18]
const imparesmMenoresQueVinte = [1,3,7,9,11,13,15,17,19]
```

Listas com números quebrados:

```
let lista = [1.4, 2.33, 6.66, 82.9]
```

AVISO: Números quebrados são escritos usando ponto (.) e não vírgula (,) em JS.

1.2.3 Objetos

Podemos criar **objetos**, que representam pacotes de informações relevantes e padronizadas. Exemplo:

```
/* Em vez de  ficar criando várias coisas,
criamos um conjunto padronizado: */

const meuEndereco = {
    cidade: 'São Paulo',
    bairro: 'Vila Pégaso',
    rua: 'Rua Algusto Mota',
    numero: 899,
    cep: '28831-321'
}

/* Posso criar pessoas, animais, endereços, tudo! Exemplo: */

const pessoa = {
    nome: 'Joana Silva Costa',
```

```
    idade: 26,
    altura: '1.78m',
    peso: '71kg',
    sexo: 'Feminino'
}
```

1.2.1 Variações dessas estruturas

Listas de listas e listas de objetos também são possíveis, e alias muito usadas:

```
/* Listas de objetos, muito úteis: */

const listaAlunosSI = [
    {
        nome: 'Joana Silva Costa',
        idade: 26,
        sexo: 'Feminino',
        curso: 'SI'
    },
    {
        nome: 'Ivan Guimarães',
        idade: 22,
        sexo: 'Masculino',
        curso: 'SI'
    },
    {
        nome: 'Paulo Souza',
        idade: 31,
        sexo: 'Masculino',
        curso: 'SI'
    }
]

/* Listas de listas: */

const listaListas = [
    [1,2,3,4,5,6],
    [1,3,4,5,6,7],
]
```

1.3 Alterando os valores das minhas variáveis:

Como o nome diz, os valores de variáveis *variam* (duhh), basta imaginar que dentro do meu sistema imaginário, minha idade depois de 1 ano irá atualizar e meu nome mudou porque eu me casei.

Exemplo abaixo de alteração de variáveis:

```
/* 1. Inventei meu nome e idade: */
let meuNome = 'Ivan Guimarães dos S. da Silva'
let minhaIdade = 22
console.log(`Seu nome: ${meuNome} | Sua idade: ${minhaIdade}`)

// Irá mostrar seu nome e idade na tela.

/* 1. Mudei meu sobrenome e minha idade: */
meuNome = 'Ivan Guimarães Rocha Sousa'
minhaIdade = 23
console.log(`Seu nome: ${meuNome} | Sua idade: ${minhaIdade}`)

// Seu nome e idade irão aparecer  diferentes pois o sistema os mudou.
```

1.4 Alterando os valores de uma constante:

Constantes diferente de variáveis não podem ser diretamente alteradas, no máximo suas subpropriedades. Mas por quê? Bem, imagine que você está fazendo um processo bem importante sobre a validação do seu CPF no banco, e de repente alguém mal intencionado muda esse processo para outra coisa. Isso é um problema, constantes garantirão ao menos que esse processo que você criou sempre será um processo desse tipo X.

Exemplo:

```
const meuNascimento = '25/04/1999'
meuNascimento = '18/10/1994'
// VAI DAR ERRO pois minha idade é uma coisa que
// representa algo inalterável. Meio filosófico mas fará
// sentido...
```

Futuramente irei mostrar outro exemplo que justifica constantes.

2. Funções: O que são e para que servem?

Função é um processo padronizado criado por nós programadores. Seu propósito é automatizar tarefas grandes.

Exemplo de função:

```
const aluno1 = { nome: 'Manoela', nota: 8.5 }
const aluno2 = { nome: 'Paulo', nota: 3.5 }

/* Função para saber se meu aluno passou:  */
function saberSeOAlunoPassou(aluno) {
    for (let caracteristica in aluno) {
        if (caracteristica == 'nota') {
            if (aluno[caracteristica] >= 6)
                console.log('Aluno passou!')
```

```
            else
                console.log('Aluno reprovado!')
        }
    }
}

saberSeOAlunoPassou(aluno1)
saberSeOAlunoPassou(aluno2)
```

Elas existem justamente para eu não ficar repetindo algo toda hora, como no exemplo abaixo:

```
const aluno1 = { nome: 'Manoela', nota: 8.5 }
const aluno2 = { nome: 'Paulo', nota: 3.5 }

/* Processo 1 para avaliar aluno: */
for(let carac in aluno1) {
    if (carac == 'nota') {
        if (aluno1[carac] >= 6)
            console.log('Aluno passou!')
        else
            console.log('Aluno reprovado!')
    }
}

/* Processo 2 para avaliar aluno: */
for(let carac in aluno2) {
    if (carac == 'nota') {
        if (aluno2[carac] >= 6)
            console.log('Aluno passou!')
        else
            console.log('Aluno reprovado!')
    }
}

/* Se eu tivesse 100 alunos isso seria horrível.  */
```

2.1 Arrow Functions

Arrow functions são funções normais, porém com uma sintaxe diferente, e sem o *hoisting*. Explicando de forma bem resumida, isso significa:

"*Se você criou um **processoB** que acontece logo após um **processoA**, **escreva essa função processoB depois de ter escrito a função processoA primeiro.***"

Exemplo de arrow function:

```
function somar(a,b) {
    return a + b
}
```

```
/* Arrow Function */
const somar2 = (a,b) => {
    return a + b
}

console.log('Soma 01: ' + somar(10, 15))
console.log('Soma 02: ' + somar2(18, 85))
```

2.2 Voltando ao conceito de constante

Como eu já havia dito, nessas horas **constantes impedem que padrões importantes para nós sejam destruídos. Em alguns senários, talvez destruídos até de propósito por alguém.**

Exemplo de processo que eu não quero que mude nunca:

```
let alunos = [
    { nome: 'Vitória', media: 8.0 },
    { nome: 'Pedro', media: 2.3 },
    { nome: 'Maria', media: 5.0 },
    { nome: 'Lucio', media: 10.0 },
]

/* Avalia quem passará e quem irá repetir de ano: */

const situacaoAlunos = (listaAlunos) => {
    let reprovados = []
    let aprovados = []

    for (let aluno of listaAlunos) {
        for (let caract in aluno) {
            if (caract == 'media') {
                if (aluno[caract] >= 6)
                    aprovados.push(aluno)
                else
                    reprovados.push(aluno)
            }
        }
    }

    return { aprovados, reprovados }
}


console.log(situacaoAlunos(alunos))
```

Caso minha função não fosse uma constante, alguém poderia modificar o processo de avaliação de alunos e seria um desastre. No exemplo abaixo a minha função *somar* é arruinada. Diferente de *somar2*.

Exemplo:

```
> let somar = (a,b) => { return a + b }
< undefined
> somar(2,2)
< 4
> somar = null
< null
> somar
< null
> somar(2,2)
✖ ▶Uncaught TypeError: somar is not a function                VM409:1
      at <anonymous>:1:1
> const somar2 = (a,b) => { return a + b }
< undefined
> somar2(2,2)
< 4
> somar2(2,6)
< 8
> somar2 = null
✖ ▶Uncaught TypeError: Assignment to constant variable.       VM653:1
      at <anonymous>:1:8
```