**Lógica e Estrutura de Dados com JavaScript**

**Aula 01**

O que é JS?

* Linguagem de programação interpretada cirada em 1995
* Tipagem dinâmica fraca
* Multiparadigma
* Evolução constante (ES6+): Praticamente uma vez por ano é lançado uma revisão/atualização da linguagem

Duas interessantes extensões a serem instaladas no VSCode

* ESLint: Encontra e corrige problemas de sintaxe nos códigos JS
* Prettier: Formatador de código para garantia de um padrão de código organizado

Ctl + j abre o terminal do VSCode

Ctrl + shif + p abrimos a janela de pesquisa do VSCode

**Tipos Primitivos**

* Number
* String
* Boolean
* Null
* Undefined

Os tipos primitivos são os tipos mais básicos de uma linguagem, onde poderemos manipular sem dificuldade todos eles.

O number será qualquer representação de número que nós iremos ter

A string será para palavras, frases, letras e alfanuméricos

O boolean tem um conceito binário 0 ou 1, ligado ou desligado, true ou false ele é usado como uma flag, ele será como um indicador para dizer se algo está ligado ou desligado, verdadeiro ou falso.

O null é representante do nulo, então quando queremos representar algo como nulo, iremos utilizar dessa estrutura.

O undefined é o que não está definido, como criamos uma variável e essa variável está sem valor, ela vai partir de undefined, já que não foi estipulado um valor.

**Expressões Aritméticas**

* Soma (+)
* Subtração (-)
* Multiplicação (\*)
* Divisão (/)
* Potência (\*\*)
* Resto (%)

A sintaxe de comentários do JS é um // (para uma única linha)

Para definirmos prioridades só podemos realizar essa operação com a utilização de []

**Expressões Lógicas**

* And (&&): Para ser verdadeiro, ambos precisam ser verdadeiros
* Or (||): Para ser verdadeiro, ao menos um precisa ser verdadeiro
* Not (!): É a negação, ou seja, not true ou not false, seu papel é inverter o papel do booleano.

**Precedência:**

O operador AND tem precedência sobre o operador Or.

**Operador Extra**

* Concatenar Strings (+)
  + String + Number

Se colocarmos “hello” + 21 o resultado será hello21, o JS transforma o número em string na concatenação

**Atribuição variável e constante**

* Declaração de variáveis e constantes: Cria um espaço na memória onde pode ser armazenado valores para reuso posteriormente
  + Variável: valor pode ser alterado no futuro
  + Constante: Após declaração (e atribuição) o valor não pode ser alterado

O rótulo é o nome da variável onde está armazenada na caixinha da memória

A atribuição é um operador denotado pelo símbolo de igual. Ele guarda um valor no espaço reservado na memória.

**Padrões de código**

* CamelCase: Isso é o CamelCase >>> (somaEntreDoisNumeros)
* Nome Semântico: Variável ter um nome que faça sentido

O CamelCase tem esse nome, pois a sua estrutura lembra a coluna do camelo.

**Exercícios:**

1. Instale o VSCode e o NodeJS no seu computador (Instale o ESLint e o Prettier também)
2. Crie um arquivo ex\_aula\_01.js e nele realize os seguintes exercícios:
   1. Imprima no console a multiplicação de 42 por -3.14
   2. Atribua a uma constante a soma das strings “Otter “ e “wise” e imprima no console seu valor
   3. Atribua a uma variável o resultado da operação AND entre os valores booleanos true e false e imprima seu valor no console
   4. Atribua a uma variável o resultado da operação OR entre os valores booleanos true e false e imprima seu valor no console
   5. Imprima no console o resultado da operação NOT o valor true

**Aula 02**

**Sumário**

* Condicionais
* Operadores de comparação
* Função (Regular)
* Escopo
* Exercícios

**Condicionais:**

* Se atendida uma determinada condição, a execução do código pode tomar um caminho diferente
* Instruções condicionais no JS:
  + IF
  + IF com ELSE e ELSEIF
  + Switch Case

O IF precisa ter uma condição lógica ou aritmética para que ele possa ter efeito.

**Operadores de comparação**

* Igual (==) e estritamente igual (===)
* Diferente(!=) e estritamente diferente (!==)
* Menor (<) e menor ou igual (<=)
* Maior (>) e maior ou igual (>=)

Boa prática é sempre utilizarmos o estritamente

**Função Regular**

* Pode-se isolar trechos de códigos em **funções** para reuso posteriormente
* Funções podem ter **parâmetros**, que são os valores de entrada da função
* Funções podem ter um retorno, que é o valor de saída da função
* Funções são definidas e posteriormente são chamadas
* Na definição da função deve ser definido seu **nome**, seus **parâmetros** e o **bloco de código** que será executado quando ela for chamada
* Chamada da função é quando o bloco de código será de fato **executado** recebendo seus argumentos e retornando seu resultado
* Parâmetros e Argumentos

Function soma(a, b) {

Return a + b

}

O “a” e o “b” são parâmetros da função, por estarem dentro do ()

É o contexto para realizar alguma operação dentro do bloco de código.

Essa função tem dois parâmetros que retornam a soma desses parâmetros

Para chamarmos essa função realizamos a chamada da soma e seus argumentos em ()

Logo: soma(3,5)

Aqui chamamos a função soma e somamos ela através dos argumentos 3 e 5 (a e b)

**Função Regular parte II**

* Funções podem, ou não, ter um retorno **explícito**
* Funções podem, ou não, ter **parâmetros**
* Qualquer valor válido no JS pode ser **passado como argumento** na chamada de uma função:
  + String, Number, Boolean, null, undefined
  + Constantes e variáveis
  + Expressões
* Da mesma forma, qualquer valor válido no JS pode ser **retornado** em uma função.

>> Funções sem parâmetro

Function getHello() { return ‘Hello World’}

>> Funções sem retorno explícito

Function helloWorld() { console.log(‘Hello world’)}

Function print(value) {console.log(value)}

Toda função terá um retorno, mesmo que ela não apareça, então quando não temos um retorno explícito, é natural que o retorno seja undefined

O return undefined não nos interessa e não iremos trabalhar com ele

**Escopo**

* Espaço onde as variáveis, constantes e funções são acessíveis
* São delimitados por chaves
* O próprio arquivo JS tem seu escopo global

Exercicios

1. Crie uma função chamada checkSignal que possui um parâmetro e imprime no console
   1. “x é positivo” se o número for positivo;
   2. “x é negativo” se o número for negativo
   3. X deve ser substituído pelo número passado como argumento
   4. Realizar uma chamada da função checkSignal passando o número -99 e outra com o número 42
2. Crie uma função chamada double que possui um parâmetro (que será um número) e retorna o dobro desse número. Chame essa função para o número 5 e imprima o resultado no console
3. Crie uma função chamada calculate que possui três parâmetros. O primeiro é uma string e os outros dois, números. A string será uma das seguintes opções “soma”, “subtrai”, “multiplica”, os dois números. Exemplo: calculate(“soma”, 3,5) deve retornar 8. Após, imprima no console quatro chamadas da função calculate, uma com cada uma das operações e números de sua escolha