# Módulo 02

## Aula 08

**JavaScript** é uma linguagem de programação, ou seja, uma maneira de escrever uma sequência de ações que serão interpretadas pelo processador.

Geralmente o JavaScript fica no final do código, seguindo uma regra de performance, para que primeiro seja carregado o conteúdo do site e depois os *scripts*.

O código JavaScript geralmente é executado no **navegador** através de um motor de JavaScript que já vem incluso nele mesmo. Mas também é possível executar esse JavaScript em outros ambientes, por exemplo por meio do **NodeJS** podemos executar fora do navegador, em um servidor por exemplo.

### Variáveis

São valores alocados em algum espaço na memória. Podem ser de três tipos: **var**, **let** ou **const**. Sendo o primeiro antigo e não mais recomendado.

* **let** – cria uma variável cujo valor pode ser alterado
* **const** – cria uma variável cujo valor é constante

As variáveis não têm explicitamente tipos dos dados (*integer*, *string*, *boolean*), por isso dizemos que é uma **linguagem não tipada**. Essa funcionalidade é adicionada futuramente pelo **TypeScript**. Os tipos de dados são inferidos implicitamente durante a execução.

* **string** – é identificada através de aspas duplas (“ “) ou aspas simples (‘ ‘)
* **number** – é simplesmente um número inteiro ou decimal, sendo as casas decimais separadas por ponto final (.)
* **boolean** – só assume dois valores: *true* ou *false*
* **undefined** – representa uma variável com valor que ainda não foi definido
* **null** – é a ausência de valor

Os tipos de dados complexos, como *object* e *array*, são representados entre chaves (**{ }**) ou colchetes (**[ ]**) respectivamente, e são implementados por baixo dos panos utilizando ponteiros (estruturas que apontam para o lugar da memória onde os dados estão armazenados).

* **object** – possui sempre uma chave e um valor (a chave é usada para apontar para o valor)
* **array** – é uma sequência de valores

Para acessar o valor de um objeto, utilizamos o **ponto** (.) seguido do nome da chave que contém o valor desejado. Para acessar o array, basta usar o nome do array seguido do número do índice do valor a ser acessado entre **chaves** (lembrando que começa do zero, então, por exemplo, o terceiro valor é acessado pelo número 2).

Exemplos: objeto.cor; array[5];

Mesmo que objetos e arrays sejam definidos como constantes, o valor do seu conteúdo ainda **pode ser alterado**. O que não pode ser alterado é o tipo da própria variável que contém o tipo complexo.

O uso de **ponto e vírgula** ao final de uma linha é opcional.

### Operadores

Podemos realizar operações entre valores ou variáveis.

Atribuição:

* **=** atribuição

Aritméticos:

* **+** adição
* **-** subtração
* **\*** multiplicação
* **/** divisão

Lógicos:

* || or (ou)
* **&&** and (e)
* **!** not (não)

Comparação:

* **==** igualdade (não considera o tipo)
* **===** igualdade estrita (considera o tipo)
* **!=** desigualdade (não considera o tipo)
* **!==** desigualdade estrita (considera o tipo)

Não podemos fazer comparações entre tipos complexos (arrays e objetos), o conteúdo sempre será diferente pois estão armazenando o endereço de memória dos valores. Para verificar se o conteúdo de um objeto mudou, o Angular e o React fazem *shallow comparison*, ao invés de *deep comparison* (valor a valor).

### Funções

É uma sequência de código que executa uma funcionalidade, e pode ser reaproveitado em outros trechos do código. Assim evitamos a redundância de escrever o mesmo código mais de uma vez.

A função pode ser declarada com a palavra **function** seguida dos ardumentos a serem recebidos entre **parênteses** e por fim o código a ser executado entre **colchetes**. Para retornar o valor utilize a palavra **return**.

A função ainda pode ser **anônima**, ou seja, sem ter um nome propriamente dito, sendo este atribuído ao invés disso a uma variável. Também não é necessário que as funções sejam declaradas no topo do código, porque o JavaScript executa um processo chamado **hoisting** para reorganizar o código na execução.

Esta em voga representar as funções como **arrow functions**, que é simplesmente uma representação mais simples e direta. Há uma pequena diferença entre esta e a função normal, quando ao caso de uso do **this**.

Exemplo de arrow function: const funcao = (x) => x + 2;

Os **argumentos** são parâmetros recebidos pela função que serão utilizados para determinar qual será o resultado daquela função. Podem ser opcionais ou obrigatórios; com valores padrão, ou não. O valor padrão é definido através de um operador de atribuição (=) na própria declaração da função.

Muitas funções já vêm incluídas no JavaScript prontas para o uso. Exemplos são as funções de cálculos matemáticos, manipulação de arrays e strings. Algumas funções de strings são: **slice**, **replace**, **replaceAll**, **toUpperCase**, **toLowerCase**, **indexOf**, **includes**.

### Condicionais

São estruturas usadas para comparar valores.

* **if** – “se esta condição for verdade, faça isso...”
* **else if** – “senão, se esta outra condição for verdade, faça isso...”
* **else** – “senão, faça isso...”

Uma maneira de escrever o *if...else* em uma só linha é por meio do **operador ternário**.

### Exercício 04

Faça uma função que calcule o preço de um terreno que tenha 5m de frente, 20m de profundidade e custe R$6.350/m².

### Exercício 05

Faça uma função que calcule a hipotenusa do triângulo retângulo, recebendo por parâmetro o valor dos seus catetos. Teste seus valores com lados de 9 e 12 (teorema de pitágoras).

### Exercício 06

Faça uma função que retorne se vale a pena ou não abastecer com álcool. A função deve receber como parâmetros o valor do álcool e da gasolina.