Aula 3 - Funções e Estruturas de Controle

aTip Learn - Lógica de Programação com JavaScript

Objetivos da Aula

- Utilizar estruturas de controle if, else e elseif.
- Criar e utilizar funções em JavaScript.

Estruturas de Controle

As estruturas de controle são utilizadas para controlar o fluxo de execução de um programa, permitindo que você tome decisões com base em condições específicas. Quando escrevemos um algoritmo ele é executado de forma linear, uma linha após a outra, porém, em muitos casos precisamos que o programa tome decisões com base em determinadas condições. Para isso utilizamos as estruturas de controle que permitem que você determine quais partes do seu código devem ou não ser executadas.

Estrutura de Controle if (se)

A estrutura de controle `if` é utilizada para executar um bloco de código **se** uma determinada condição for verdadeira. A sintaxe da estrutura de controle `if` é a seguinte:

```
if (condição) {
    // Bloco de código a ser executado se a condição for verdadeira
}
```

A *condição* é uma expressão lógica que pode ser avaliada como verdadeira ou falsa. Se a condição for verdadeira, o bloco de código dentro das chaves `{}` será executado, caso contrário, o bloco de código será ignorado.

```
let idade = 18;

if (idade >= 18) {
    console.log('Você é maior de idade');
}
```

Nesse exemplo a variável idade armazena o valor 18, e a condição `idade >= 18` verifica se a idade é maior ou igual a 18. Como essa condição é verdadeira (`idade` é igual a 18), a nossa condição é verdadeira e o bloco de código dentro das chaves `{}` é executado, exibindo a mensagem `'Você é maior de idade'` no console.

Condições Verdadeiras e Falsas Em JavaScript, diferente de muitas outras linguagens, uma condição não precisa resultar no valor booleano `true` ou `false`, qualquer valor pode ser avaliado como verdadeiro ou falso. Valores que são avaliados como falsos são chamados de *falsy* e valores que são avaliados como verdadeiros são chamados de *truthy*.

Esse conceito é motivo de muitas confusões para quem está começando na linguagem JavaScript, por isso é importante entender quais valores são avaliados como verdadeiros e quais são avaliados como falsos, e sempre que possível utilizar condições que resultem em valores booleanos. Abaixo estão listados os valores que são avaliados como falsos em JavaScript:

- `false`
- `0`
- `''` (string vazia)
- `null`
- `undefined`
- `NaN` (Not a Number, valor resultante de operações matemáticas inválidas)

Condições Compostas Além das condições simples como vimos no exemplo anterior, é possível criar condições compostas utilizando os operadores lógicos `&&` (E) e `||` (OU). Ou seja, uma condição composta é um conjunto de condições que precisam ser satisfeitas para que o bloco de código seja executado. O resultado da combinação de duas condições é determinado pelo operador lógico utilizado. Veja o exemplo abaixo:

```
let idade = 18;
let possuiCNH = true;

if (idade >= 18 && possuiCNH) {
    console.log('Você pode dirigir');
}
```

Nesse exemplo, a condição `idade >= 18 && possuiCNH` verifica se a idade é maior ou igual a 18 **e** se a pessoa possui CNH. Se ambas as condições forem verdadeiras, o bloco de código dentro das chaves `{}` será executado, exibindo a mensagem `'Você pode dirigir'` no console. Perceba que não é necessário comparar a variável `possuiCNH` com `true`, pois o próprio valor da variável já é avaliado como verdadeiro.

```
let hora = 14;
let estaChovendo = true;

if (hora >= 18 || estaChovendo) {
  console.log('Ligue os faróis');
}
```

Nesse exemplo, a condição `hora >= 18 || estaChovendo` verifica se a hora é maior ou igual a 18 **ou** se está chovendo. Se qualquer uma das condições for verdadeira, o bloco de código dentro das chaves `{}` será executado, exibindo a mensagem `'Ligue os faróis'` no console.

Estrutura de Controle else (senão) A estrutura de controle `else` é utilizada para executar um bloco de código **se** a condição do `if` for falsa. Ou seja, a estrutura `else` sempre é utilizada em conjunto com a estrutura `if`, e é executada quando a condição do `if` é falsa. A sintaxe da estrutura de controle `else` é a seguinte:

```
if (condição) {
    // Bloco de código a ser executado se a condição for verdadeira
} else {
    // Bloco de código a ser executado se a condição for falsa
}
```

```
let idade = 17;

if (idade >= 18) {
    console.log('Você é maior de idade');
} else {
    console.log('Você é menor de idade');
}
```

Nesse exemplo, a variável idade armazena o valor 17, e a condição `idade >= 18` verifica se a idade é maior ou igual a 18. Como essa condição é falsa (`idade` é igual a 17), o bloco de código dentro das chaves `{}` do `else` é executado, exibindo a mensagem `'Você é menor de idade'` no console.

Estrutura de Controle else if (senão se) A estrutura de controle `else if` é utilizada para executar diferentes blocos de código com base em diferentes condições exclusivas. A estrutura `else if` é utilizada entre as estruturas `if` e `else`, e é executada se a condição do `if` for falsa e a condição do `else if` for verdadeira. A sintaxe da estrutura de controle `else if` é a seguinte:

```
if (condição1) {
    // Bloco de código a ser executado se a condição1 for verdadeira
```

```
} else if (condição2) {
    // Bloco de código a ser executado se a condição2 for verdadeira
} else {
    // Bloco de código a ser executado se nenhuma das condições anteriores for verdadeira
}
```

```
let idade = 21;

if (idade >= 18) {
    console.log('Você é maior de idade');
} else if (idade >= 16) {
    console.log('Você é adolescente');
} else {
    console.log('Você é criança');
}
```

Nesse exemplo, a variável idade armazena o valor 21, e a condição `idade >= 18` verifica se a idade é maior ou igual a 18. Como essa condição é verdadeira, o bloco de código dentro das chaves `{}` do `if` é executado, exibindo a mensagem `'Você é maior de idade'` no console. Se a condição do `if` for falsa, a condição do `else if` é verificada, e se for verdadeira, o bloco de código dentro das chaves `{}` do `else if` é executado, exibindo a mensagem `'Você é adolescente'` no console. Se nenhuma das condições anteriores for verdadeira, o bloco de código dentro das chaves `{}` do `else` é executado, exibindo a mensagem `'Você é criança'` no console.

Pergunta: Se você trocar a ordem das condições `idade >= 16` e `idade >= 18`, o que aconteceria?

Estruturas de Controle Aninhadas As estruturas de controle `if`, `else if` e `else` podem ser aninhadas, ou seja, podemos utilizar uma estrutura de controle dentro de outra. Isso é útil quando precisamos verificar múltiplas condições de forma mais complexa. Veja o exemplo abaixo:

```
let idade = 21;
let possuiCNH = true;

if (idade >= 18) {
    if (possuiCNH) {
        console.log('Você pode dirigir');
    } else {
        console.log('Você não pode dirigir');
    }
} else {
    console.log('Você é menor de idade');
}
```

Nesse exemplo, a variável idade armazena o valor 21 e a variável possuiCNH armazena o valor `true`. A primeira condição `idade >= 18` verifica se a idade é maior ou igual a 18, se essa condição for verdadeira, a estrutura de controle `if` aninhada é executada. Dentro dessa estrutura de controle aninhada, a condição `possuiCNH` verifica se a pessoa possui CNH, se essa condição for verdadeira, o bloco de código dentro das chaves `{}` é executado, exibindo a mensagem `'Você pode dirigir'` no console. Se a condição `possuiCNH` for falsa, o bloco de código dentro das chaves `{}` do `else` é executado, exibindo a mensagem `'Você não pode dirigir'` no console. Se a condição `idade >= 18` for falsa, o bloco de código dentro das chaves `{}` do `else` é executado, exibindo a mensagem `'Você é menor de idade'` no console.

Funções

Imagine que você está desenvolvendo um programa que precisa calcular a média das 4 notas bimestrais de um aluno. Você provavelmente desenvolveria algo parecido com o código abaixo:

```
let nota1 = 7;
let nota2 = 8;
let nota3 = 6;
let nota4 = 9;

let media = (nota1 + nota2 + nota3 + nota4) / 4;
```

Agora imagine que você precisa repetir esse mesmo processo para os 40 alunos de uma turma, ou quem sabe para centenas de alunos de uma escola. Você teria que repetir esse mesmo código várias vezes, o que rapidamente deixaria de ser prático, seu código ficaria muito extenso e difícil de manter. Quando um problema (ou uma parte do problema) se apresenta de forma repetitiva, em geral a melhor solução é criar uma função que resolva esse problema, e então poderemos utilizar essa função sempre que precisarmos resolver esse problema.

Uma função é basicamente um bloco de código identificado por um nome, o qual pode ter suas próprias entradas e saídas. Assim como podemos utilizar o valor de uma variável em diferentes partes do código apenas referenciando-a pelo nome, também podemos chamar uma função em diferentes partes do código utilizando seu nome. A sintaxe para declarar uma função é a seguinte:

```
function nomeDaFuncao(parametro1, parametro2, ...) {
    // Bloco de código a ser executado
    return valorDeRetorno;
}
```

- `nomeDaFuncao`: É o nome da função, que deve ser único e significativo, ou seja, deve representar o que a função faz. Geralmente utilizamos verbos para nomear funções, como `calcularMedia`, `exibirMensagem`, `validarUsuario`, etc.
- `parametro1, parametro2, ...`: São os parâmetros da função, que são valores que a função espera receber para realizar suas operações, ou seja, são as entradas da função. Os parâmetros são utilizados dentro da função da mesma forma que as variáveis. Uma função pode não ter nenhum parâmetro, ou pode ter quantos parâmetros forem necessários.
- `return valorDeRetorno`: A instrução `return` é utilizada para retornar um valor da função para o local onde a função foi chamada. O `valorDeRetorno` é o valor que a função irá retornar, e pode ser qualquer tipo de dado, como um número, uma string, um objeto, um array, etc. Em JavaScript uma função pode pode não retornar nenhum valor, ou retornar apenas um valor.

Vamos reescrever o código que calcula a média das notas dos alunos em uma função:

```
function calcularMedia(nota1, nota2, nota3, nota4) {
    let media = (nota1 + nota2 + nota3 + nota4) / 4;
    return media;
}
```

Nesse exemplo estamos declarando uma função chamada `calcularMedia` que recebe 4 parâmetros, que são as notas dos alunos. Dentro da função estamos calculando a média das notas e retornando esse valor. Para utilizar essa função, basta chamá-la passando os valores das notas como argumentos:

```
let mediaAluno1 = calcularMedia(7, 8, 6, 9);
console.log(mediaAluno1); // 7.5
let mediaAluno2 = calcularMedia(5, 6, 7, 8);
console.log(mediaAluno2); // 6.5
```

Nesse exemplo estamos chamando a função `calcularMedia` duas vezes, passando as notas dos alunos como argumentos. A função irá calcular a média das notas e retornar esse valor, que será armazenado nas variáveis `mediaAluno1` e `mediaAluno2`, e então exibimos esses valores no console. Quando a função é chamada, os parâmetros informados na chamada da função são atribuídos às variáveis dentro da função, essa atribuição é feita de acordo com a ordem dos parâmetros. Ou seja na primeira chamada da função `calcularMedia(7, 8, 6, 9)`, o valor `7` é atribuído à variável `nota1`, o valor `8` é atribuído à variável `nota2`, e assim por diante.

A linha `return media;` é responsável por retornar o valor da variável `media` para o local onde a função foi chamada. Assim, quando a função é chamada e o cálculo da média é feito, o valor da média é retornado e armazenado na variável `mediaAluno1` ou `mediaAluno2`, que então pode ser utilizado em outras partes do código.

Algumas funções não recebem parâmetros, e outras funções não retornam valores. Por exemplo, a função abaixo exibe uma mensagem no console, mas não recebe nenhum parâmetro e não retorna nenhum valor:

```
function exibirMensagem() {
    console.log('Olá, mundo!');
}
```

Se tentarmos receber um valor dessa função, atribuindo-o à variável mensagem, o valor da variável será `undefined`, pois a função não retorna nenhum valor:

```
let mensagem = exibirMensagem(); // Exibe 'Olá, mundo!' no console
console.log(mensagem); // Exibe 'undefined' no console
```

O mesmo ocorre com uma função que espera receber um parâmetro, mas em sua chamada não é passado nenhum valor para esse parâmetro. Nesse caso se tentarmos utilizar o parâmetro no corpo da função, o valor será `undefined`:

```
function exibirMensagem(nome) {
    console.log('Olá, ' + nome + '!');
}

exibirMensagem(); // Exibe 'Olá, undefined!' no console
```

- Documentação sobre Funções

Escopo da Função

Quando declaramos uma função, os parâmetros dessa função e as variáveis declaradas no corpo da função somente são acessíveis dentro do corpo dessa função, ou seja, elas possuem escopo local. Escopo é o contexto onde uma variável ou função é acessível. Se tentarmos acessar uma variável declarada dentro de uma função fora do corpo da função, o JavaScript irá retornar um erro, pois essa variável não é acessível fora do escopo da função.

```
function exibirMensagem() {
    let mensagem = 'Olá, mundo!';
    console.log(mensagem);
}

exibirMensagem(); // Exibe 'Olá, mundo!' no console
console.log(mensagem); // Erro: ReferenceError: mensagem is not defined
```

Nesse exemplo, a variável `mensagem` é declarada dentro da função `exibirMensagem`, e por isso ela só é acessível dentro do corpo da função. Quando tentamos acessar a variável `mensagem` fora do corpo da função, o JavaScript retorna um erro, pois essa variável não é acessível fora do escopo da função.

- Documentação sobre Escopo

Escopo Global

Por outro lado, variáveis declaradas fora do corpo das funções são acessíveis em qualquer parte do código, ou seja, elas possuem escopo global e podem ser utilizadas tanto dentro quanto fora das funções. Variáveis declaradas fora do corpo das funções são chamadas de variáveis globais.

```
let mensagem = 'Olá, mundo!';
function exibirMensagem() {
```

```
    console.log(mensagem);
}

exibirMensagem(); // Exibe 'Olá, mundo!' no console
console.log(mensagem); // Exibe 'Olá, mundo!' no console
```

Desafios

1. Crie uma função chamada `obterMaior` que recebe 2 números como parâmetros e retorna o maior deles. Utilize essa função para encontrar o maior número entre 4 e 7.
2. Crie uma função chamada `mostrarOrdemCrescente` que recebe 3 números como parâmetros e exibe esses números em ordem crescente. Utilize essa função para exibir os números 5, 3 e 7 em ordem crescente.
3. Crie uma função chamada `cumprimentar` que recebe um nome como parâmetro e exibe a mensagem `'Olá, [nome]!'`. Caso o nome não seja informado a função deve exibir a mensagem `'Você não me disse o seu nome!'`. Chame essa função passando o nome `'João'` como parâmetro e sem passar nenhum nome como parâmetro.
4. Crie uma função chamada `nomeCompleto` que recebe 2 parâmetros, `nome` e `sobrenome`, e retorna o nome completo. Utilize essa função oara obter o nome completo de `'José'` e `'Souza'`.
5. Crie uma função chamada `verificarEstoque` que recebe 3 parâmetros, `quantidade`, `baixo` e `crítico`, e retorna `'Estoque normal'` se a quantidade for maior que o valor de `baixo`, `'Estoque baixo'` se a quantidade for menor ou igual ao valor de `baixo` e maior que o valor de `crítico`, e `'Estoque crítico'` se a quantidade for menor ou igual ao valor de `crítico`. Utilize essa função para verificar o estoque de 100 unidades, com valor baixo de 50 unidades e valor crítico de 20 unidades.

Desafio Extra

Você sabia que alguns tipos na linguagem JavaScript possuem suas próprias funções (chamadas de métodos)? Por exemplo, o tipo de dados `string` possui métodos que permitem manipular ou obter informações sobre strings. Um desses métodos é o método `toUpperCase`, que retorna a string em que ele foi chamado convertida para letras maiúsculas. Por exemplo, o código abaixo exibe a string `'OLÁ, MUNDO!'` no console:

```
let mensagem = 'Olá, mundo!';
let mensagemMaiuscula = mensagem.toUpperCase();
console.log(mensagemMaiuscula); // 'OLÁ, MUNDO!'
```

Você pode encontrar uma lista completa de métodos disponíveis para o tipo `string` no site da Mozilla Developer Network na lista `Métodos de Instância`. Clicando sobre o nome do método você consegue ver uma descrição detalhada do método, exemplos de uso e a compatibilidade com os diferentes navegadores.

Nesse desafio você deve desenvolver uma função chamada `abreviarNome` que recebe 2 parâmetros, `nome` e `sobrenome`, e retorna as iniciais do nome e do sobrenome em letras maiúsculas. Exemplo:

```
let joseSouza = abreviarNome('josé', 'souza');
console.log(joseSouza); // Deve exibir 'J.S.'

let mariaSilva = abreviarNome('maria', 'silva');
console.log(mariaSilva); // Deve exibir 'M.S.'
```

Para concluir esse desafio você vai precisar utilizar o método `toUpperCase` que exemplificamos acima, além de algum outro método que lhe permita obter o primeiro caractere de cada `string`. Existem diversos métodos que podem lhe ajudar nessa tarefa, pesquise na documentação da Mozilla Developer Network para encontrar o método que achar mais adequado.