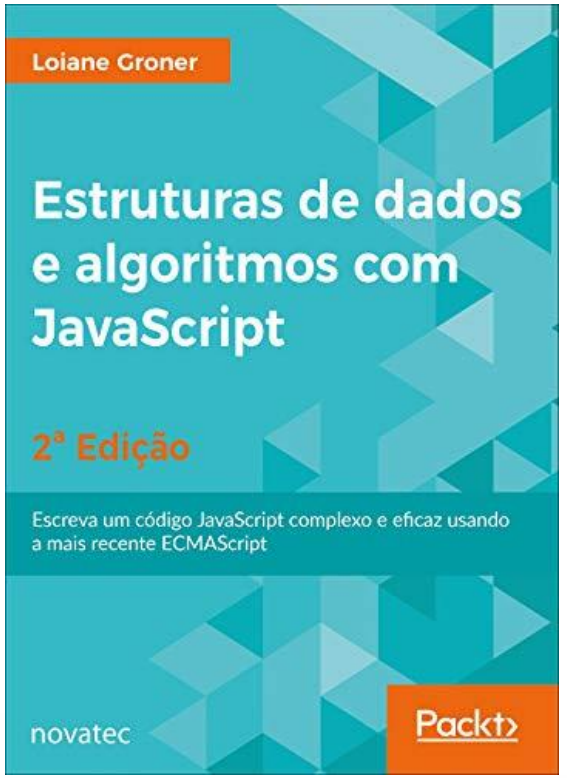
Estruturas de Controle em JavaScript

Referências Bibliográficas:

GRONER, Loiane. **Estruturas de dados e algoritmos com JavaScript**: Escreva um código JavaScript complexo e eficaz usando a mais recente ECMAScript. 2ª ed. São Paulo: Novatec, 2019.



SIF005 - Estrutura de Dados
Prof. Dr. Anderson - anderson.sena@edu.iesb.br

Estruturas de controle

- A linguagem JavaScript tem um conjunto de estruturas de controle semelhante ao de outras linguagens, como C e Java.

- Instruções condicionais são tratadas com **if...else** e **switch**.

- Laços são tratados com as construções **while**, **do...while** e **for**.

```
1   var num = 1;
2   if (num === 1) {
3       console.log('num is equal to 1');
4   }
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRu
num is equal to 1
```

```
6   var num = 0;
7   if (num === 1) {
8       console.log('num is egual to 1');
9   } else {
10      console.log('num is not equal to 1, the value of num is: ' + num);
11  }
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
num is not equal to 1, the value of num is: 0
```


Groner, 2019, pág. 41

Instruções condicionais

- A instrução **if..else** também pode ser representada por um **operador ternário**, veja o exemplo:

```
if (num === 1) {
    num--;
} else {
    num++
}
```

- Essa instrução também pode ser representada assim:

```
// Essa instrução também pode ser representada assim:
(num === 1) ? num-- : num++;
```

Instruções condicionais

- Além do mais, se tivermos várias expressões, podemos usar **if...else** diversas vezes para executar blocos de código diferentes, de acordo com condições distintas, assim:

```
24  var month = 5;
25  if (month === 1) {
26      console.log('January');
27  } else if (month === 2) {
28      console.log('February');
29  } else if (month === 3) {
30      console.log('March');
31  } else {
32      console.log('Month is not January, February or March');
33  }
34
```

```
PROBLEMS 3   OUTPUT   TERMINAL   DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
Month is not January, February or March
```


Groner, 2019, pág. 41

Instruções condicionais

Se a condição que estivermos avaliando for a mesma que a anterior podemos usar a instrução **switch.**

Palavras reservadas:

- case: *verifica se o valor é igual (==) da variável switch entre parêntesis.*
- break: *interrompe a sequência;*
- default: *executa por padrão, caso nenhuma opção case seja* **true**.

```
36  var month = 5;
37  switch (month) {
38      case 1:
39          console.log('January');
40          break;
41      case 2:
42          console.log('February');
43          break;
44      case 3:
45          console.log('March');
46          break;
47      default:
48          console.log('Month is not January, February or March');
49          break;
50  }
51
```

PROBLEMS 3 OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
Month is not January, February or March
```

- Os laços de repetição são usados com frequência quando trabalhamos com **arrays**.
- O laço **for** é exatamente igual ao das linguagens C e de Java.
- É constituído de um contador de laço que, em geral, recebe um valor numérico; em seguida, a variável é comparada com outro valor (*o script dentro do laço for será executado enquanto essa condição for verdadeira*):

```
52    for (var i = 0; i < 10; i++) {
53        console.log(i);
54    }
```

PROBLEMS 3 OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-
0
1
2
3
4
5
6
7
8
9
```

Somente um lembrete

- Antes que alguém me pergunte: "**Professor, porquê o comando console.log() não imprime os valores na mesma linha?**"
- A instrução **console.log( )** tem a finalidade de fazer **debug**.
- Ela não serve para mostrar a saída do seu programa, ***não é para isso que ela foi concebida!***
- Para mostrar algum texto na saída, considerando que está usando **javascript**, há diversas maneiras que você pode usar.
- Para começar, tente alguma das seguintes:

```
1.
var saida = 'Texto qualquer';
document.write(saida);

2.
var saida = 'Texto qualquer';
alert(saida);
```

- O próximo laço que veremos é o **while**.
- O bloco de código dentro do laço while será executado enquanto a condição for verdadeira.

```
70   var i = 0;
71   while (i < 10) {
72       console.log(i);
73       i++;
74   }
75
```

PROBLEMS 3 OUTPUT TERMINAL DEBUG CON

```
[Running] node "c:\Users\admin\Desktop
0
1
2
3
4
5
6
7
8
9
```

- O laço **do...while** é muito parecido com o laço **while**.
- A única diferença é que, no laço **do...while**, a condição é avaliada depois de o bloco de código ter sido executado.
- O laço **do...while** garante que o bloco de código seja executado pelo menos uma vez.

```
78  var i = 0;
79  do {
80      console.log(i);
81      i++;
82  } while (i < 10);
83
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-
0
1
2
3
4
5
6
7
8
9
```

Groner, 2019, pág. 43

- As **funções** são muito importantes quando trabalhamos com JavaScript.

- O código da figura, mostra a sintaxe básica de uma função.

- Ela não tem argumentos (*parâmetros*) nem a instrução **return** (*valor de retorno*):

ed-js > js > JS estruturasDeControle.js > ...

```
1
2   function sayHello() {
3       console.log('Hello');
4   }
5
6   // Para executar esse código, basta
7   //usar a chamada da função pelo seu nome
8   sayHello();
9
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tem
Hello
```

Groner, 2019, pág. 44

- Você pode usar quantos argumentos quiser.
- Mas nesse caso, apenas o primeiro argumento será usado pela função, o segundo será ignorado, veja:

ed-js > js > JS estruturasDeControle.js > ...

```
1  // Também podemos passar argumentos (parâmetros) para uma função
2  function output(text) {
3      console.log(text);
4  }
5
6  // nesse caso, apenas o primeiro parâmetro será
7  // usado pela função; o segundo será ignorado
8  output('Boa noite!', 'Olá tudo bem!');
9
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
Boa noite!
```

- Uma função também pode devolver um valor:

ed-js > js > JS estruturasDeControle.js > ...

```
1   // Também podemos passar argumentos (parâmetros) para uma função
2   function output(text) {
3       console.log(text);
4   }
5
6   // Essa função calcula a soma de dois números
7   // especificados e devolve o resultado.
8 > function sum(num1, num2) {···
10  }
11
12  // chamando a execução da função
13  var result = sum(2, 3);
14  output(result);
15
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
5
```


Groner, 2019, pág. 44

Funções

- Codificando uma função para cálculo simples de nota de aluno, diretamente no código de uma página HTML:

- Para executar, basta clicar com o botão direito no arquivo .html e optar por abrir (*executar*) com o plugin do servidor web LiveServer.

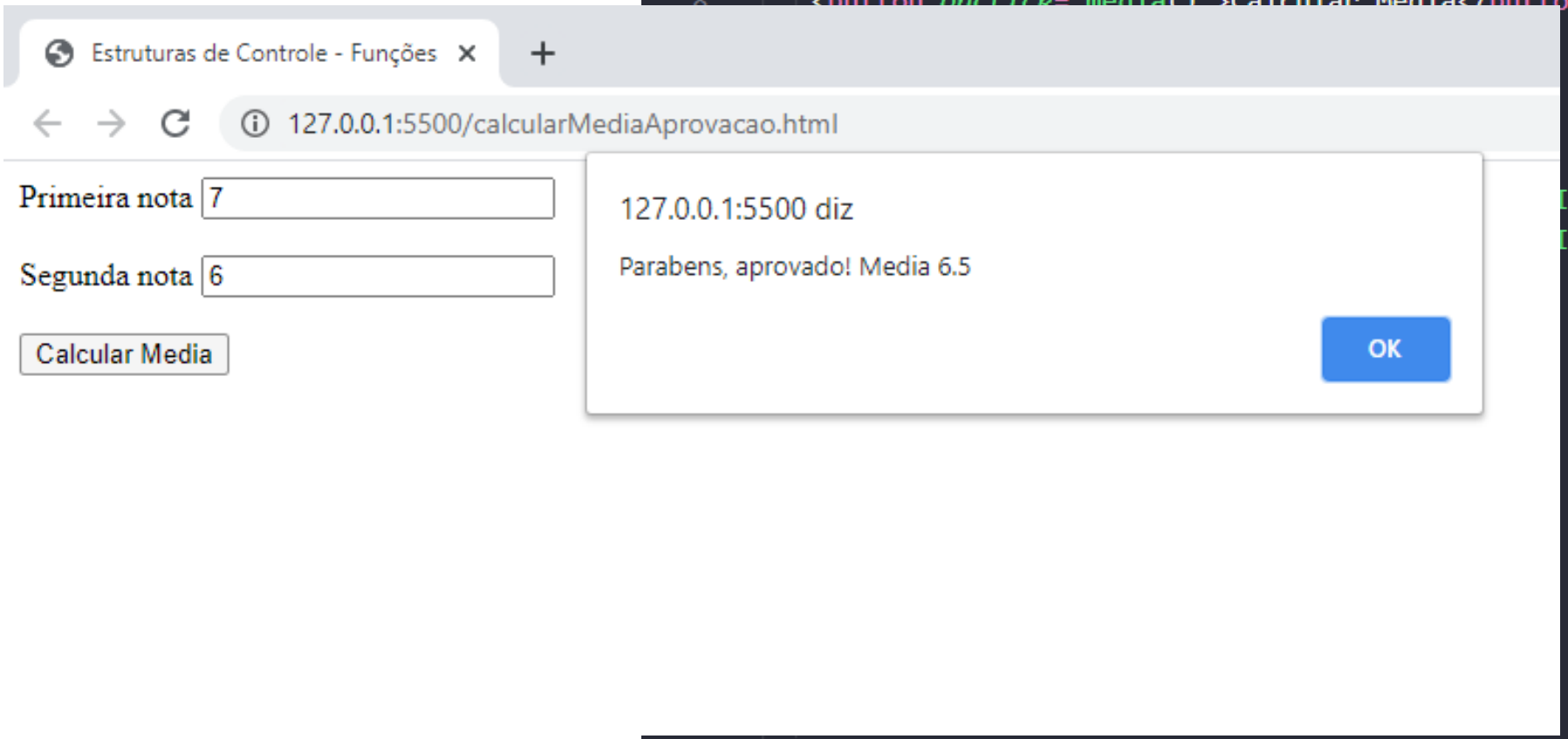
ed-js > calcularMediaAprovacao.html > html > head

```
<!DOCTYPE html>
<html>
 <head>
   <title>Estruturas de Controle - Funções</title>
   Primeira nota <input id="nota1" type="number"> <br /><br />
   Segunda  nota <input id="nota2" type="number"> <br /><br />

   <button onclick="media()">Calcular Media</button>

   <script type="text/javascript">

    function media(){
     var nota1 = parseFloat(document.getElementById("nota1").value);
     var nota2 = parseFloat(document.getElementById("nota2").value);

     var media = (nota1 + nota2)/2 ;

     if(media >= 5)
      if(media==10)
       alert("Uau! Aprovado com Louvor!");
      else
       alert("Parabens, aprovado! Media "+media);
     else
      alert("Reprovado!")

    }
 </script>

```

Funções

- Para executar, basta clicar com o botão direito no arquivo .html e optar por abrir (*executar*) com o plugin do servidor web LiveServer.

```
ed-js > calcularMediaAprovacao.html > html > head
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <title>Estruturas de Controle - Funções</title>
5       Primeira nota <input id="nota1" type="number"> <br /><br />
6       Segunda  nota <input id="nota2" type="number"> <br /><br />
7
8       <button onclick="media()">Calcular Media</button>
```




```
Id("nota1").value);
Id("nota2").value);
```

```
27    </script>
28
```

Programação orientada a objetos em Javascript

- Objetos em JavaScript são coleções bem simples de pares nome-valor.
- Há duas maneiras de criar (*instanciar*) um objeto simples em JavaScript.
  - A primeira é assim:

```
1
2   // criando (instanciando) com new
3   var obj = new Object();
4
```

  - A segunda é assim:

```
1
2   var obj = {};
3
```


Groner, 2019, pág. 45

Programação orientada a objetos em Javascript

- Também podemos criar um objeto completo, desta maneira:

```
1
2   // criando um objeto completo [chave, val
3   obj = {
4       name: {
5           first: 'Regiano',
6           last : 'Alves'
7       },
8       address: 'QSF 15 casa 13 - Tag. Sul'
9   };
10
11  console.log(obj.address);
12
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js
QSF 15 casa 13 - Tag. Sul
```

- Pares de [chave, valor]
  - **chave** pode ser considerada um atributo do objeto.
  - **valor** é o valor da propriedade.
- Em **POO**, um objeto é uma **instância** de uma classe.
- Uma **classe** define as **características** do objeto.

Obs.: todas as classes que criaremos serão objetos JavaScript, como **Stack**, **Set**, **LinkedList**, **Dictionary**, **Tree**, **Graph** etc.

Groner, 2019, pág. 45

Uma maneira de declarar uma classe (construtor) que representa um livro:

```
1   // Declarando uma classe (construtor) que representa um livro:
2   function Book (title, pages, isbn) {
3       this.title = title;
4       this.pages = pages;
5       this.isbn = isbn;
6   }
7   // para instanciar essa classe, podemos usar o código a seguir
8   var book = new Book('Estrutura de Dados', 406, '978-85-7522-553-0');
9   // Acessando suas propriedades
10  console.log(book.title); // exibe o título do livro
11  book.title = 'Estrutura de Dados e algoritmos com JavaScript' // atualiza o valor do título
12  console.log(book.title); // exibe o valor atualizado
13  console.log('Este livro possui: ' + book.pages + ' páginas.');
14
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
Estrutura de Dados
Estrutura de Dados e algoritmos com JavaScript
Este livro possui: 406 páginas.
```


Groner, 2019, pág. 46

- Uma classe também pode conter **funções** (**métodos**).
- Podemos declarar e usar uma **função/método**:

```
1   // Declarando uma classe (construtor) que representa um livro:
2   function Book (title, pages, isbn) {
3       this.title = title;
4       this.pages = pages;
5       this.isbn = isbn;
6   }
7   // para instanciar essa classe, podemos usar o código a seguir
8   var book = new Book('Estrutura de Dados', 406, '978-85-7522-553-0');
9   // Podemos declarar e usar uma função/método, veja:
10  Book.prototype.printTitle = function() {
11      console.log(this.title);
12  };
13  book.printTitle();
14
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
Estrutura de Dados
```

- ✓ No exemplo com prototype, a função printTitle será compartilhada entre todas as instâncias, e somente uma cópia será criada.
- ✓ Quando usamos, uma definição baseada em classe, como no exemplo, cada instância terá a sua própria cópia das funções.
- ✓ O uso do método prototype economiza memória e processamento quando se atribuir funções à instância.
- ✓ Você só pode declarar funções e propriedades **public** usando o método prototype.

Groner, 2019, pág. 46

- Também podemos declarar funções diretamente na definição da classe.

✓ Com uma definição baseada em classe, você pode declarar **funções** e **propriedades private**, e os outros métodos da classe também poderão acessá-las.

```
1   // Declarando uma classe (construtor) que representa um livro:
2   function Book (title, pages, isbn) {
3       this.title = title;
4       this.pages = pages;
5       this.isbn = isbn;
6       this.printIsbn = function() {
7           console.log('ISBN: ' + this.isbn);
8       }
9   }
10  // para instanciar essa classe, podemos usar o código a seguir
11  var book = new Book('Estrutura de Dados', 406, '978-85-7522-553-0');
12  // executando (chamando) a propriedade função da classe
13  book.printIsbn();
14
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
ISBN: 978-85-7522-553-0
```


Groner, 2019, pág. 46

Exercícios de fixação:

1. Faça um script para solicitar o nome do usuário. Exiba uma mensagem de boas vindas com o nome informado. Use o método prompt() dentro da tag <script></script> da página html:

```
var nome = prompt( 'Informe o seu nome:', '' );
```

2. Utilize uma estrutura de controle para descobrir se o valor de sua idade é par ou ímpar.

3. Faça um laço para imprimir todos os anos, do ano atual até o ano de seu nascimento.

4. Peça ao usuário para digitar idades de 10 pessoas. Exiba quantas pessoas são maior de idade (18 anos) e quantas são menores.

5. Declare as seguintes propriedades (variáveis) em uma classe aluno (nome, idade e peso), instancie um objeto com seu nome. Depois acesse e altere os valores correspondentes aos seus tipos: nome(string), idade(inteiro), peso(double). Apresente na console o conteúdo de cada propriedade do objeto.