Aula 13: Introdução a Objetos

Nesta aula apresentaremos a você o conceito de objetos na linguagem JavaScript. Você aprenderá aqui os objetos em detalhes. Entenderá conceitos de classe, instância, propriedades e métodos. Além de conhecer diversas características do objeto `String`.

Objetivos:
Nesta aula você aprenderá:
- o que são objetos;
- os conceitos de classe e instância;
- o que são propriedades e métodos;
- a criação de uma instância;
- os tipos de objeto em JavaScript: embutidos, do browser e personalizados; e
- as características e métodos do objeto `String`.

Pré-requisitos:
As aulas anteriores deste módulo, especialmente a seção que explica as regras de escopo da aula passada, além de uma revisão da Aula 1 do módulo 1.

1. Programação Orientada a Objetos

Constantemente, vêm sendo criadas técnicas para facilitar o desenvolvimento e a manutenção dos programas. Estas técnicas consistem principalmente em regras que, uma vez seguidas, agilizam e facilitam o processo de desenvolvimento. A programação orientada a objetos neste sentido é mais que uma técnica, ela busca modificar a forma como o programador vê o problema a ser solucionado, criando uma **abstração** mais próxima do mundo real do que nas linguagens de programação mais antigas.

A programação orientada a objetos vê um problema como um conjunto de entidades (objetos) que interagem. Cada entidade tem suas características próprias (atributos ou propriedades) e faz a interação com outros objetos por meio de uma interface (métodos). Ela usa um modelo de programação que "reflete" o mundo real melhor que as formas de programação mais antigas, facilitando a divisão de tarefas, escondendo a complexidade e estimulando o aproveitamento de código de programas anteriores.

Na perspectiva tradicional de resolução de problemas, primeiro se decide quais as operações (funções) que serão efetuadas, depois se pensa em quais dados estarão envolvidos. Numa perspectiva "orientada a objetos", primeiro se identifica quais as entidades envolvidas para depois pensar na interação entre elas.

2. Objetos, Propriedades e Métodos

Em linguagens de programação, um **objeto** é uma coleção de variáveis diversas (com valores de qualquer tipo) à qual se atribui um **nome**. Essas variáveis são usualmente denominadas **propriedades** (ou atributos) do objeto. Para se referir a uma propriedade de um objeto, você deve identificar o objeto seguido do operador "ponto" e o nome da propriedade. Por exemplo: imagine um objeto denominado `imagem` e que tenha as propriedades denominadas `largura` e `altura`. Neste caso, você se refere a estas propriedades escrevendo:

```
imagem.largura
imagem.altura
```

As **propriedades** dos objetos podem conter qualquer tipo de dado, incluindo referências a funções e outros objetos. Assim, você pode ter um objeto `documentos`, por exemplo, que tenha uma propriedade que seja o objeto `imagem` acima e neste caso você se referiria à `largura` como:

```
documentos.imagem.largura
```

Quando uma referência a uma função é armazenada em uma propriedade de um objeto ela é chamada de **método**. Para invocar esta função, usa-se também o nome do objeto, seguido do operador **ponto** e depois o nome do método seguido de parênteses (como em uma função). Você deve se lembrar de que isso já foi utilizado antes para invocar o método **write** do objeto **document**, como em:

```
document.write("exemplo");
```

Uma variável de um tipo de dados primitivo armazena um dado único, como um único número, uma única `string`, um único booleano. Já um objeto é um tipo composto, ele pode agregar múltiplos valores de dados e nos permite armazenar ou recuperar todos os valores pelo **nome** comum.

Resumindo, um objeto é uma forma de agrupar dados para

representar uma estrutura mais complexa. Os dados que compõem um objeto são chamados de atributos ou propriedades. Uma maneira de explicar isso em outras palavras é dizer que um objeto é uma coleção de **propriedades**, cada uma com um nome e valor. Os objetos também introduzem uma vinculação entre os dados e as operações ou métodos neles realizadas.

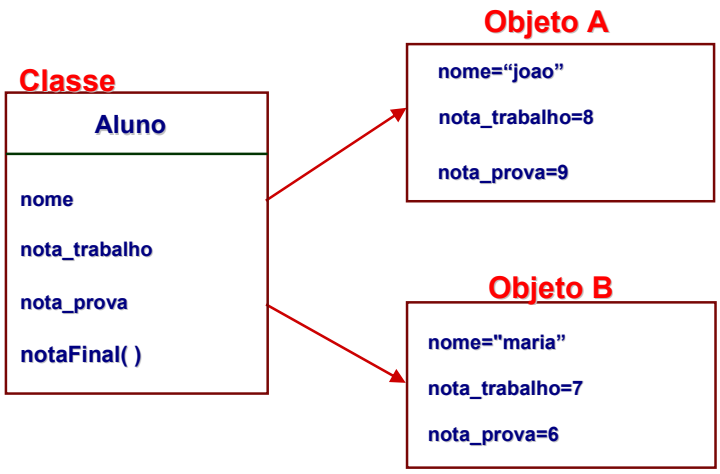
3. Entendendo Classes de Objetos

Como foi dito anteriormente, um objeto pode ter **variáveis** que descrevem o seu estado (**propriedades**) e interagem com os demais objetos por meio de funções (**métodos**). Propriedades são como variáveis da linguagem, podem ser de qualquer tipo (inclusive um outro objeto).

Uma **Classe** é como um molde a partir do qual criamos um objeto. Definir uma classe é definir as variáveis que a compõem e os métodos criados para manipulá-las. Uma vez definida uma classe é possível criar variáveis para guardar seus dados (como se fosse um tipo da linguagem). Criar um objeto é criar uma variável cujo tipo seja uma classe (chamamos isso de criar uma instância ou instanciar).

Imagine que você queira fazer um programa para "controlar" as notas de seus amigos em uma determinada matéria. Para desenvolver este programa, você possivelmente vai precisar estabelecer um vínculo entre as informações escolares de seu amigo (nota do trabalho e nota da prova) e o nome dele, manipulando todas estas informações como uma só. Neste ponto você acabou de levantar os dados de seu interesse.

Figura 13.1 - Ilustração de classe e objetos



Se você somar a esta definição o código necessário para manipular estes dados (para obter, por exemplo, a nota final), e der a este conjunto um nome (`Aluno`, por exemplo), você acaba de definir uma **classe**.

Depois de definir a classe você poderia criar objetos desta classe, criando variáveis que armazenariam os dados componentes da classe. Em linguagem mais técnica diz-se que você estaria instanciado a classe. No esquema mostrado na figura 13.1 tentamos ilustrar isso.

Uma variável para guardar um objeto é criada através da invocação de uma função especial chamada **construtor**. O construtor precisa ser executado através do operador **new**. Por exemplo, dada a classe `Aluno`, o código abaixo cria duas variáveis deste tipo.

```
a = new Aluno();
b = new Aluno();
a.nome = “joao”;
a.nota_trabalho = 8;
```

A função principal do construtor é inicializar as propriedades do objeto. Apesar da obrigatoriedade de ser sempre invocado através do operador `new`, o construtor é uma função e como tal pode receber parâmetros. No exemplo anterior, podemos pensar que não faz sentido criar um objeto `Aluno` que não tenha seu nome definido. O construtor poderia então receber um atributo `string` que inicializasse a propriedade nome. O código anterior seria então reescrito como:

```
a = new Aluno(“joão”);
b = new Aluno(“Maria”);
a.nota_trabalho = 8;
```

Se nós quiséssemos calcular a nota final teríamos de combinar o valor das propriedades `nota_trabalho` e `nota_prova`. A forma mais correta de fazer isso seria criar um método que faria este cálculo e "retornaria" para o programa o valor desejado. A vantagem de fazer isso através de um método e não pela manipulação direta das propriedades é que, se futuramente mudar a forma de cálculo da nota final (por exemplo, se passar a haver duas provas), será necessário apenas reescrever o código da função `notaFinal( )`, sem precisar modificar os pontos onde esta função estiver sendo chamada. No exemplo a seguir, poderíamos então imprimir a nota final de um aluno usando o seguinte código:

```
document.write("Nota final de ",
       a.nome," é ",a.notaFinal());
```

Vejamos outro exemplo: é possível definir qualquer círculo a partir da posição de seu centro (definido pelo par de coordenadas `xcentro` e `ycentro`) e o seu `raio`. Dado um círculo é possível desenhá-lo ou verificar se um ponto (definido pelas coordenadas `px` e `py`) está em seu interior ou não. Sendo assim, para a classe `círculo`, teríamos os atributos `xcentro`, `ycentro` e `raio` e os métodos `desenha` e `verificaSeDentro`:

Tabela 13.1 - Ilustrando propriedade e métodos

Propriedades	Métodos
xcentro ycentro raio	Desenha() verificaSeDentro(px,py)

Para ter acesso aos métodos e propriedades de um objeto utiliza-se o operador "`.`" (ponto). Assim, se definirmos um objeto `circuloPequeno` como sendo da classe `circulo`, seria possível definir ou acessar os valores do `raio` e do `centro` (ou outros atributos) através da sintaxe:

```
circuloPequeno.xcentro = 10;
circuloPequeno.ycentro = 20;
circuloPequeno.raio = 5;
document.write("Perímetro=",
            2*3.14*circuloPequeno.raio);
```

E seria possível usar os métodos, que desenham o círculo e verificam se um ponto está dentro, através da sintaxe:

```
circuloPequeno.desenha()
circuloPequeno.verificaSeDentro(15,25);
```

Resumindo: classes são como tipos de dados compostos, que definem como serão os objetos de um determinado tipo. Um método é uma função ligada diretamente a uma classe de objetos e escrita para manipular suas propriedades. Além da capacidade de criar e utilizar objetos, JavaScript pode adicionar propriedades aos objetos dinamicamente. Esta não é uma ocorrência possível em linguagens estritamente baseadas em classes. Neste sentido ela não tem a noção formal de classe de outras linguagens fortemente tipadas como C++ e Java. Diz-se que JavaScript não é uma linguagem orientada a objetos baseado em classes.

4. Tipos de Objeto de JavaScript

Em JavaScript existem 3 tipos de objetos:

- os embutidos ou predefinidos (`Array, Date, Math, String`).
- os do `browser` (`Window, Document, Navigator`).
- os personalizados ou criados pelo programador.

Durante este curso trataremos apenas dos dois primeiros tipos de objetos. Os objetos do `browser` (e o objeto `Math`) são previamente criados e já os encontramos à nossa disposição quando o programa começa a ser executado. Nos exemplos anteriores, inclusive, já foi utilizado o objeto `document` e invocado um de seus métodos (`write`). Os objetos embutidos `Date` e `Array` funcionam como um tipo de dado e é possível criar variáveis para armazená-los. Diferente porém dos outros tipos de dados, é necessário utilizar o operador `new` para fazer a inicialização do objeto antes de poder utilizá-lo:

```
data = new Date;
vetor = new Array(10);
```

A seguir são descritos e exemplificados os objetos embutidos, suas propriedades e seus métodos.

5. O Objeto String

Em computação ou em linguagem de computador, **string** é qualquer série de caracteres alfanuméricos ou palavras consecutivas que são manipuladas e tratadas como uma unidade pelo computador.

Em capítulos anteriores já se usou do objeto `String`. Este objeto embutido oferece apoio ao uso de textos, diferente de outros, não necessita ser inicializado através do operador `new`. Assim, são formas de criá-lo:

```
nome="Maria";
telefone= new String("26666666");
```

Viu-se também que é possível utilizar o operador + com o sentido de concatenação de dois objetos `String`, como já foi usado em diversas ocorrências anteriores do curso ou como no exemplo seguinte:

```
dados = nome + ":" + telefone;
```

Este objeto tem uma propriedade que não pode ser modificada (diz-se que é uma propriedade *"read-only"*: apenas para leitura) e diversos métodos descritos nas tabelas 13.1 e 13.2 que seguem. A propriedade `length` mostra o número de caracteres contidos na `string`. Todos os métodos e a propriedade são acessados usando o `operador ponto` após o nome da `string` no qual operarão.

13.2 - Propriedade do objeto embutido `String`

Propriedade	Significado
`Lenght`	Número de caracteres (ou tamanho) da `string`.

13.3 - Métodos do objeto embutido `String`

Métodos	Significado
`anchor(nome)`	Faz uma cópia da `string` entre as `tags` `<a name="nome">` e `</a>`.
`big(nome)`	Faz uma cópia da `string` entre as `tags` `<big>` e `</big>`.
`blink()`	Faz uma cópia da `string` entre as `tags` `<blink>` e `</blink>`.
`bold()`	Faz uma cópia da `string` entre as `tags` `<b>` e `</b>`.
`charAt(i)`	Retorna o caracter da `string` na posição `i` (começa por 0). Se `i` não está entre 0 e o número de caracteres da `string` , retorna uma `string` vazia.
`charCodeAt(i)`	Retorna o código Unicode do caracter na posição `i` da `string`.
`concat(s1,...)`	Retorna uma nova `string` resultante da concatenação das `s1, s2, ...` . A `string` inicial não é modificada.
`fixed()`	Faz uma cópia da `string` entre as `tags` `<TT>` e `</TT>`.
`Fontcolor(color)`	Faz uma cópia da `string` entre as `tags` `<font color = "color">` e `</font>`.
`fontsize(i)`	Faz uma cópia da `string` entre as `tags` `<font size="i">` e `</font>`, `i` pode ser de `1` a `7`, `+` ou `-` .

13.4 - Métodos do objeto embutido `String` - continuação

`fromCharCode (c1,c2,c3,...)`	Cria uma nova `string` contendo os caracteres especificados pelos códigos Unicode `c1,c2,`...
`indexOf (s, i)`	Retorna a posição da `string` `s` na `string`, começando a busca da posição `i`.
`italics()`	Faz uma cópia da `string` entre as tags `<i>` e `</i>`.
`lastIndexOf(s,i)`	Retorna a posição da `string` `s` na `string`, começando a busca da posição `i` para o início.
`link(href)`	Faz uma cópia da `string` entre as `tags` `<A HREF="href">` e `</A>`.
`replace(sesp,s)`	Faz uma busca da `substring` `sesp` e a substitui por `s`. Aceita propriedades especiais na busca.
`search(s)`	Retorna a posição na `string` da `substring` `s`.
`slice(i,f)`	Retorna a `substring` entre as posições `i` e `f`.
`small()`	Faz uma cópia da `string` entre as tags `<small>` e `</small>`.
`split(s)`	Retorna um `Array` com cada palavra da `string` usando a `string s` como separador.
`strike()`	Faz uma cópia da `string` entre as `tags` `<strike>` e `</strike>`.
`sub()`	Faz uma cópia da `string` entre as `tags` `<sub>` e `</sub>`.
`substr(i,n)`	Retorna `n` caracteres da `string` a partir do caracter na posição `i`.
`substring (i,f)`	Retorna a `string` entre o i-esimo e o f-esimo caracter.
`sup()`	Faz uma cópia da `string` entre as `tags` `<sup>` e `</sup>`.
`toUpperCase()`	Converte uma `string` para imprimi-la em maiúsculas (não modifica a variável).
`toLowerCase()`	Converte uma `string` para imprimi-la em minúsculas (não modifica a variável).

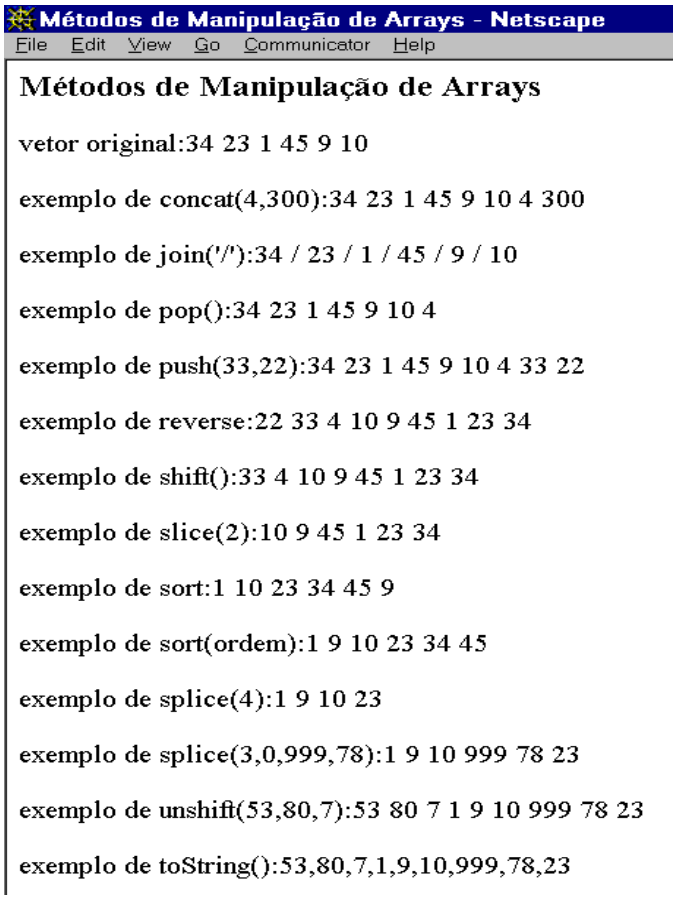
O código que segue exemplifica o uso de cada um destes métodos .

```
<HTML>
 <HEAD>
  <TITLE>Exemplificando os métodos de
anipulação de strings</TITLE>
 </HEAD>
 <BODY>
  <SCRIPT LANGUAGE="javascript">
  texto = "O Ivo viu a uva";
  document.write('<h2>Na string "' +
    texto + '" tem-se:</h2>');
  document.write('Tamanho = ' +
    texto.length + '<br>');
  document.write('anchor()= ' +
    texto.anchor("inicio") + '<br>');
  document.write('big()= ' + texto.big()
    + '<br>');
  document.write('blink()= ' +
    texto.blink() + '<br>');
  document.write('bold()= ' +
    texto.bold() + '<br>');
  document.write('charAt (10) = "' +
    texto.charAt(10) + '"<br>');
  document.write('charAt (20) = "' +
    texto.charAt(20) + '"<br>');
  document.write('charCodeAt (10) = "' +
    texto.charCodeAt(10) + '"<br>');
  document.write('concat(" verde ",
    "para"," vinho!") = "' +
    texto.concat(" verde"," para",
    "vinho!") + '"<br>');
  document.write('fixed() = "' +
    texto.fixed() + '"<br>');
  document.write('fontcolor("red") = "' +
    texto.fontcolor("red") + '"<br>');
  document.write('fontsize(5) = ' +
    texto.fontsize(5) + '"<br>');
  var s=String.fromCharCode(101,108,97);
  document.write('fromCharCode(101,108,97)="'
    + s + '"<br>');
  document.write('indexOf ("viu") = ' +
    texto.indexOf("viu") + "<br>");
  document.write('italics() = ' +
    texto.italics() + "<br>");
  document.write('lastIndexOf ("u") = ' +
    texto.lastIndexOf("u") + "<br>");
  document.write('link()= ' +
    texto.link("#inicio") + '<br>');
  document.write('replace("uva","Eva")= '
    + texto.replace("uva","Eva") +
    '<br>');
  document.write('search("uva")= ' +
    texto.search("uva") + '<br>');
  document.write('slice(3,10)= ' +
```

```
    texto.slice(2,10) + '<br>');
  document.write('small() = ' +
    texto.small() + "<br>");
  document.write('split(" ")= ' +
    texto.split(" ") + '<br>');
  document.write('strike() = ' +
    texto.strike() + "<br>");
  document.write('sub() = ' + texto.sub()
    + "<br>");
  document.write('substr (6, 5) = "'+
    texto.substr(6, 5) + '"<br>');
  document.write('substring (6, 11) = "'+
    texto.substring(6, 11) + '"<br>');
  document.write('sup() = ' + texto.sup()
    + "<br>");
  document.write('toLowerCase () = "' +
    texto.toLowerCase() + '"<br>');
  document.write('toUpperCase () = "' +
    texto.toUpperCase() + '"<br>');
  document.write('texto = "' + texto +
    '"<br></b>');
  </SCRIPT>
 </BODY>
</HTML>
```

Para entender exatamente o que cada método faz e como usá-lo, observe sua descrição na tabela 13.4, depois veja como ele é empregado no trecho de código que o utiliza e observe o resultado obtido com sua execução na figura que segue.

Figura 13.2 - Ilustração do objeto `String`



6. Cadeia de Escopo de Variáveis

Você já deve ter notado que existem muitas semelhanças na linguagem JavaScript entre variáveis e propriedades dos objetos. Na realidade não há uma diferença fundamental entre ambas e pode-se dizer que em JavaScript variáveis são fundamentalmente a mesma coisa que propriedades de objetos.

6.1. O Objeto Global

Quando o interpretador da linguagem inicia, uma das primeiras coisas que faz, antes de executar qualquer código de JavaScript, é criar um **objeto global.** O objeto global está no topo da cadeia de escopo. As propriedades deste objeto global são as variáveis globais dos programas JavaScript. Quando você define uma variável global, o que está realmente fazendo é definir uma propriedade do objeto global. Além disso, todas as funções predefinidas e

propriedades do ambiente JavaScript também são propriedades do objeto global.

6.2. Variáveis Locais e o Objeto Chamado

Você deve estar se perguntando a esta altura se as variáveis globais são propriedades de um objeto global especial, o que são então as variáveis locais? Elas também são propriedades de um objeto. Este objeto é conhecido como o **objeto chamado** (***call object***). Este objeto tem um período de "vida" menor que o objeto global, mas serve para o mesmo propósito. Enquanto uma função está sendo executada, os argumentos e as variáveis locais são armazenados como propriedade deste objeto, o **objeto chamado**. O uso de um outro objeto para as variáveis locais é o que possibilita a linguagem fazer o valor de variáveis locais se sobrepor ao valor de variáveis globais de mesmo nome.

6.3. Contexto de Execução de Javascript

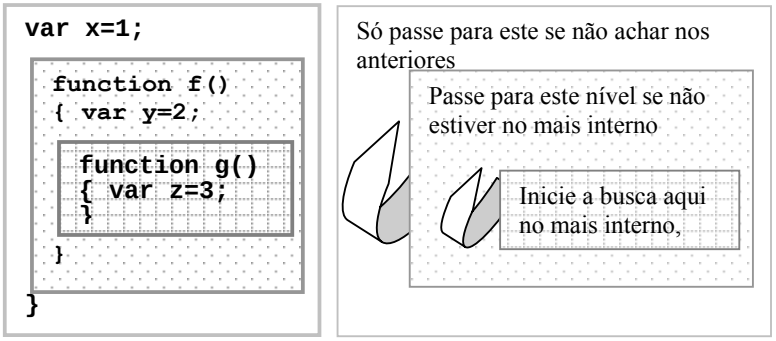
Cada vez que o interpretador da linguagem inicia a execução de uma função, ele cria um novo contexto de execução desta função. Um contexto de execução é obviamente o contexto no qual cada pedaço de código é executado. Uma parte importante do contexto é o objeto no qual as variáveis são definidas. Cada função em JavaScript roda em seu próprio contexto, que por sua vez chama o objeto no qual as variáveis locais são definidas. Um código que não faz parte de nenhuma função, roda em um contexto de execução que usa o objeto global para definição de variáveis.

Na primeira vez que discutimos a noção de escopo de variáveis, a noção básica apresentada era que variáveis globais tinham escopo global e as variáveis declaradas dentro de funções tinham escopo local. Se uma função é definida dentro de outra, as variáveis declaradas na função mais interior têm um escopo mais local. Agora que se falou que as variáveis globais são propriedades de um objeto global e que as locais são propriedades de um objeto especial, o **objeto chamado**, podemos aprofundar este conceito ampliando-o para diversos contextos.

Cada contexto de execução tem um escopo em cadeia associado a ele. Este escopo é uma lista ou uma cadeia de objetos. Quando o código precisa do valor de uma variável `x`, para obtenção deste valor, inicia um processo chamado de **resolução do nome da variável**. Neste processo ele começa procurando pelo objeto mais interno da cadeia. Se neste

objeto é encontrada a propriedade de nome x, seu valor é usado. Se neste nível não há a propriedade x, a variável é procurada no próximo nível da cadeia, e assim se movendo dos níveis mais interiores para os exteriores até encontrar a variável procurada. A variável só é indefinida se não for achada até a busca chegar ao topo da cadeia, ou seja, ao objeto global. A figura 13.3 ilustra essas formas de obtenção do valor de uma variável em um escopo em cadeia.

Figura 13.3 - Ilustrando escopo de variáveis



Exercícios:

1. Utilizando a estrutura do código exemplo, faça um pequeno programa utilizando todos os métodos do objeto `String`.

2. Faça um pequeno programa, utilizando o objeto `Strings` e os seus métodos, que transforme para maiúsculas e depois para minúsculas tudo o que o usuário digitar. Escreva neste programa uma função que permita ao usuário encontrar uma palavra que ele fornecer.

Resumo:

Nesta aula você foi apresentado aos objetos da linguagem JavaScript. Conheceu em detalhes as características da classe `String` de objetos predefinidos. Na próxima aula continuaremos neste assunto estudando outros objetos que permitirão a criação de programas muito versáteis.

Auto-avaliação:

Quantos conceitos novos nesta aula! Se algum ponto não ficou muito claro, releia-o antes da próxima aula para continuarmos a entender esta poderosa ferramenta que são os objetos.