

DESIGN DE INTERFACE PARA WEB

AULA 8 – UTILIZANDO OBJETOS EM JAVASCRIPT

WALTER TRAVASSOS SARINHO

@WALTEROPROFESSOR

WALTER.TRAVASSOS@UNIPE.EDU.BR



Mapa de Estudos

AULA 01

Arquitetura de Sistemas Web e sua evolução em camadas



AULA 03

Formulários HTML

AULA 02

Introdução ao HTML, principais TAGS

AULA 04

Introdução ao JavaScript.
Funções declaradas e eventos HTML

AULA 05

Tipos de dados JavaScript.
Principais operadores e introdução ao DOM com getElementById

AULA 06

Estruturas de decisão, repetição e arranjos em JavaScript.

Avaliação A1

Mapa de Estudos

AULA 07

Funções: `addEventListener`, `createElement()`, `appendChild()` e `removeChild()`.
Hoisting e expressões de função

AULA 09

Template Strings, `for in` e `typeof`

AULA 08

Orientação a Objetos de forma prototipada (antes do ECMA Script 6)
Criação de classes (depois do ECMA Script 6)

AULA 10

API JSON e API Web Storage

AULA 11

JQuery

AULA 12, 13, 14...

NodeJS

Avaliação A2



JS

Aula de Hoje

- Utilização de objetos em JavaScript.
- Compreender o uso de gets e sets.
- Praticar.

Introdução

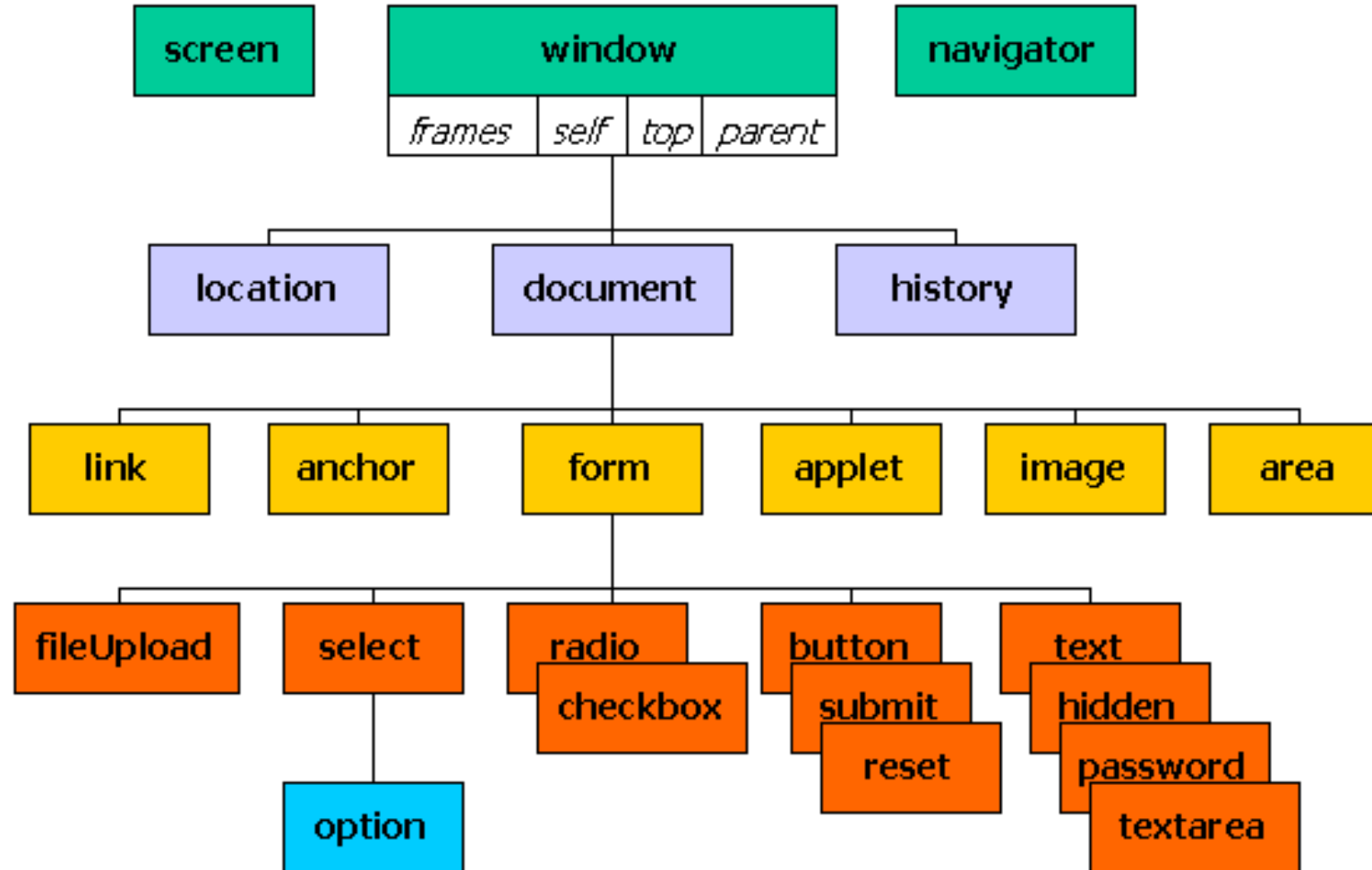
- A linguagem JavaScript é projetada com base em um **simples paradigma orientado a objeto**.
- Um objeto é uma coleção de propriedades, e uma propriedade é uma associação entre um nome (ou chave) e um valor, por exemplo, **cor : “vermelho”**.
- Um valor de propriedade também pode ser uma função, que é então considerada um **método** do objeto.
- Além dos objetos que são pré-definidos no browser, você pode definir seus próprios objetos.



Objetos são como Objetos

- Em JavaScript um objeto é uma entidade independente, com propriedades e tipos.
- Compare-o com uma xícara, por exemplo. Uma xícara é um objeto, com propriedades.
- Uma xícara tem uma cor, uma forma, peso, um material de composição, etc.
- Da mesma forma, objetos em JavaScript podem ter propriedades, que definem suas características.

Existem objetos pré-definidos no browser



Outros Objetos JavaScript

JavaScript representa boa parte dos seus recursos a partir de objetos:

- Strings
- Datas
- Cálculos Matemáticos
- Expressões Regulares
- Arrays
- Funções

Objetos JavaScript

- Um objeto JavaScript pode ser comparada a uma variável que pode conter vários valores ao mesmo tempo:

```
const pessoa = { nome : "Walter", idade : 42, cor : "Vermelho" };
```

- Suas propriedades (ou atributos) são escritas na forma **nome :** **valor**, separados por vírgula.

Dot Notation

- A "dot notation" é uma convenção usada em linguagens de programação, incluindo JavaScript, para acessar propriedades de objetos. Ela é chamada de "dot notation" porque utiliza um ponto (.) para acessar as propriedades de um objeto.
- Em JavaScript, objetos são estruturas de dados que podem conter propriedades e métodos. Para acessar uma propriedade de um objeto usando a "dot notation", você escreve o nome do objeto seguido de um ponto e o nome da propriedade que deseja acessar.

Objetos JavaScript

- Assim, uma propriedade de um objeto JavaScript pode ser acessada a partir da sintaxe:

nomeDoObjeto.nomeDaPropriedade

- Exemplo:

```
const pessoa = {  
  nome: "Walter",  
  idade: 42  
};
```

```
confirm(pessoa.nome + " tem " + pessoa.idade + " anos");
```

No entanto, a título de segurança, podemos criar métodos gets e sets

- Um método é uma função invocada por um objeto.
- Os métodos gets e sets são os “métodos padrões” para acessar (get) e modificar (set), respectivamente, as variáveis de instância de uma classe.
- Os métodos sets não retornam valor. Eles mudam apenas o conteúdo da variável de instância de um objeto.
- Já os métodos gets retornam alguma coisa. Possuem o comando return.

Método get

Além de propriedades, um objeto JavaScript também pode ter métodos (funções contidas na própria variável, acessíveis a partir dela):

```
const pessoa = {  
  nome: "Walter",  
  idade: 42,  
  getNome: function() {  
    return this.nome;  
  }  
};
```

Método get

- Um método de um objeto JavaScript pode ser acessada a partir da sintaxe:

```
nomeDoObjeto.nomeDoMetodo()
```

- Exemplo:

```
const pessoa = {  
  nome: "Walter",  
  idade: 42,  
  getNome: function() {  
    return this.nome;  
  }  
};  
confirm(pessoa.getNome() + " gosta de vermelho");
```


Método set

O método set altera o valor da variável de instância.

```
const pessoa = {  
  nome: "Walter",  
  idade: 42,  
  getNome: function() {  
    return this.nome;  
  },  
  setNome: function(nom) {  
    this.nome = nom;  
  }  
};
```

get e set da variável nome

```
1 ▼ var pessoa = {  
2     nome: "Walter",  
3 ▼   getNome: function() {  
4       return this.nome;  
5   },  
6 ▼   setNome: function(nom) {  
7       this.nome = nom;  
8   }  
9 };  
10  
11 pessoa.setNome(prompt( ));  
12 confirm(pessoa.getNome( ) + " gosta de vermelho");
```

Complete os espaços em branco

```
const pessoa = {  
  matricula: 123,  
  getMatricula: function() {  
    return _____;  
  },  
  
  setMatricula: function(mat) {  
    _____;  
  }  
};
```

Exercícios

Exercício 1

Crie uma aplicação HTML + JavaScript que permita o lançamento de informações para uma agenda eletrônica. Crie um documento HTML onde o usuário poderá inserir as seguintes informações: **Nome** e **telefone**. No JavaScript crie um objeto (**contato**) com os atributos nome e telefone, além de métodos para setar e obter dados dos mesmos (baseados nas informações inseridas pelo usuário) e imprimir as informações da agenda no navegador.

Utilize as funções `addEventListener` e `appendChild` em sua resposta.

nome:

telefone:

Resposta:

nome:

telefone:

Resposta:
Contato salvo

nome:

telefone:

Resposta:
Nome: WALTER TRAVASSOS SARINHO | Telefone: 8388888888

Resposta sem Separação de Responsabilidade

Usando o evento onclick

Exercício 1 – Resposta HTML

```
<body>  
  Nome: <input type="text" id="nome"><br>  
  Telefone: <input type="text" id="fone"><br>  
  <button value="salvar" onclick="salvar()">Salvar</button>  
  <button value="mostrar" onclick="mostrar()">Mostrar</button>  
  <br>Resposta:  
  <div id="resposta"></div>
```

Exercício 1 – Resposta JavaScript

```
1 ▼ var agenda = {  
2     nome: "",  
3     telefone: "",  
4 ▼   getNome: function() {  
5       return this.nome;  
6     },  
7 ▼   setNome: function(nom) {  
8       this.nome = nom;  
9     },  
10 ▼  getFone: function() {  
11      return this.telefone;  
12    },  
13 ▼  setFone: function(fone) {  
14      return this.telefone = fone;  
15    }  
16 };
```

```
18 ▼ function salvar() {  
19     agenda.setNome (document.getElementById("nome").value);  
20     agenda.setFone (document.getElementById("telefone").value);  
21     document.getElementById("resposta").innerHTML =  
22         "Contato salvo";  
23 }  
24 ▼ function mostrar() {  
25     document.getElementById("resposta").innerHTML =  
26         "Nome: "+agenda.getNome() + " | Telefone: "+agenda.getFone();  
27 }
```

Resposta com Separação de Responsabilidade

Usando `addEventListener`

Exercício 1 – Resposta HTML

```
11 <body>
12     Nome <input type="text" id="nome"><br>
13     Telefone: <input type="text" id="telefone"><br>
14     <button id="salvar">Salvar</button>
15     <button id="mostrar">Mostrar</button><br>
16     Resposta: <div id="resposta"></div>
17     Contato: <div id="contato"></div>
```

Exercício 1 – Resposta Javascript - Objeto

```
19 <script>
20 const contato = {
21     nome : "",
22     telefone : "",
23     setNome: function(nom) {this.nome = nom;},
24     getNome: function() {return this.nome;},
25     setFone: function(fon) {this.fone = fon;},
26     getFone: function() {return this.fone;}
27 };
```

Exercício 1 – Resposta Javascript - salvar

```
29     const btnSalvar = document.getElementById("salvar");
30
31     btnSalvar.addEventListener("click", function() {
32         contato.setNome(document.getElementById("nome").value);
33         contato.setFone(document.getElementById("telefone").value);
34
35         const divResposta = document.getElementById("resposta");
36         const paragrafo = document.createElement("p");
37
38         paragrafo.textContent = "Contato salvo";
39         divResposta.appendChild(paragrafo);
40     });
```


Exercício 1 – Resposta Javascript - mostrar

```
42  const btnMostrar = document.getElementById("mostrar");
43
44  btnmostrar.addEventListener("click", function() {
45      const divContato = document.getElementById("contato");
46      const paragrafo = document.createElement("p");
47
48      paragrafo.textContent = "Nome: " + contato.getNome() + " Telefone: " + contato.getFone();
49      divContato.appendChild(paragrafo);
50
51  });
52
53
54  </script>
```

Funções Construtoras

Constructor Functions

Constructor Functions

- Até agora, estamos construindo objetos de forma individual.
- Se quisermos criar um novo objeto com valores diferentes, temos que criar o novo objeto e alterar o valor de suas propriedades. No entanto, nesse formato, o objeto original pessoa está tendo seus valores alterados, não estamos criando um novo objeto a partir do original.
- Quando fazemos isso, pessoa1 e pessoa terão os mesmos valores que pessoa2.

JS script.js

```
1  const pessoa = {  
2    nome: "",  
3    idade: 0  
4  }  
5  
6  const pessoa1 = pessoa;  
7  pessoa1.nome = "Walter";  
8  pessoa1.idade = 42;  
9  
10 const pessoa2 = pessoa;  
11 pessoa2.nome = "Zé";  
12 pessoa2.idade = 20;  
13  
14 console.log(pessoa);  
15 console.log(pessoa1);  
16 console.log(pessoa2);
```

```
Object {nome: "Zé", idade: 20}  
Object {nome: "Zé", idade: 20}  
Object {nome: "Zé", idade: 20}
```

Funções Construtoras

- As funções construtoras em JavaScript são funções especiais que são usadas com a palavra-chave **new** para criar objetos.
- Quando uma função é invocada com **new**, ela se comporta como um construtor e retorna um novo objeto.
- Isso é útil para criar múltiplas instâncias de um tipo de objeto comum, compartilhando métodos e propriedades.

Exemplo

Olá, meu nome é Ana e tenho 30 anos.
Olá, meu nome é João e tenho 25 anos.

JS script.js

```
1 // Definição de uma função construtora para criar objetos Pessoa
2 function Pessoa(nome, idade) {
3     this.nome = nome;
4     this.idade = idade;
5     this.saudacao = function() {
6         console.log(`Olá, meu nome é ${this.nome} e tenho ${this.idade} anos.`);
7     };
8 }
9 // Criando instâncias de objetos usando a função construtora
10
11 const pessoa1 = new Pessoa('Ana', 30);
12 const pessoa2 = new Pessoa('João', 25);
13
14 pessoa1.saudacao(); // Saída: Olá, meu nome é Ana e tenho 30 anos.
15 pessoa2.saudacao(); // Saída: Olá, meu nome é João e tenho 25 anos.
```

No exemplo...

- A função Pessoa atua como uma **função construtora**.
- Quando usada com **new**, ela cria e retorna um novo objeto com as propriedades definidas dentro dela (no caso, nome e idade).
- A função **saudacao** é um método que exibe uma saudação usando os detalhes da pessoa.
- Um aspecto fundamental das funções construtoras é o uso do **this**. Dentro da função construtora, **this** se refere ao novo objeto que está sendo criado.
- Ao atribuir propriedades e métodos a **this**, você está definindo-os no novo objeto.

Protótipos

prototype

Protótipos

- O sistema de protótipos é a **base do modelo de herança** e serve para compartilhar propriedades e métodos entre objetos.
- Cada objeto em JavaScript tem uma **referência interna** para outro objeto chamado de **protótipo**.
- Quando uma propriedade de um objeto é acessada, o interpretador **procura primeiro no próprio objeto** e, se não encontrar, **busca essa propriedade nos protótipos**. Se não encontrar no protótipo, continua a busca na "**cadeia de protótipos**", até encontrar a propriedade ou chegar ao fim da cadeia.

Exemplo

JS script.js

```
1 function Game() {  
2   this.pulou = function() {  
3     console.log('o personagem pulou');  
4   }  
5   this.deitou = function() {  
6     console.log('o personagem deitou');  
7   }  
8 }  
9  
10 Game.prototype.correu = function() {  
11   console.log('o personagem correu');  
12 }  
13  
14 const novoJogo = new Game();  
15 novoJogo.pulou();  
16 novoJogo.deitou();  
17 novoJogo.correu();
```


o personagem pulou
o personagem deitou
o personagem correu

Assim...




- Os protótipos em JavaScript fornecem uma maneira eficiente de compartilhar funcionalidades entre objetos e são a base do sistema de herança do JavaScript.
- Isso permite criar uma estrutura de objetos com propriedades e métodos comuns, mantendo um código mais eficiente em termos de memória e compartilhamento de funcionalidades.

ChatGPT responde



Deixa ver se entendi uma coisa do javascript e do uso de construtores em sua orientação a objetos. Quer dizer que não utilizamos o conceito de "new class", utilizamos apenas funções contrutoras para criar objetos? 



Exatamente, em JavaScript, até a versão ECMAScript 5 (ES5), não havia uma sintaxe nativa para definir classes, como é comum em outras linguagens de programação orientadas a objetos. Em vez disso, JavaScript usa funções construtoras para criar objetos e simula a orientação a objetos por meio de protótipos.   

Utilizamos Funções Construtoras para criar objetos

- Até a introdução do ECMAScript 6 (ES6), JavaScript não possuía uma sintaxe de classe semelhante a outras linguagens de programação orientadas a objetos, como Java ou Python.
- No entanto, as funções construtoras serviam como uma maneira de criar estruturas de objetos reutilizáveis.

Por exemplo:

Funções Construtoras com Protótipos

```
1 function Pessoa(nome, idade) {  
2     this.nome = nome;  
3     this.idade = idade;  
4 }  
5  
6 Pessoa.prototype.saudacao = function() {  
7     console.log(`Olá, meu nome é ${this.nome} e tenho ${this.idade} anos.`);  
8 };  
9  
10 const pessoa1 = new Pessoa('Ana', 30);  
11 const pessoa2 = new Pessoa('João', 25);  
12  
13 pessoa1.saudacao(); // Saída: Olá, meu nome é Ana e tenho 30 anos.  
14 pessoa2.saudacao(); // Saída: Olá, meu nome é João e tenho 25 anos.
```

É uma Orientação a Objetos real?

- O ECMAScript 6 (ES6) introduziu uma nova sintaxe de classe para JavaScript, que é uma forma mais clara e semelhante à de outras linguagens.
- Ainda assim, por baixo dos panos, essa sintaxe de classe ainda se baseia no protótipo do JavaScript. Isso significa que, por baixo, as classes em ES6 ainda estão os protótipos.
- Portanto, as classes em JavaScript são uma camada de abstração mais amigável para os desenvolvedores que desejam trabalhar com o paradigma de orientação a objetos, mas por trás disso, a herança prototípica continua a ser o sistema fundamental do JavaScript.

Classes

Classes

- As classes foram introduzidas no ECMAScript 6 (ES6).
- Elas são uma forma de criar objetos e definir métodos, propriedades e construtores que representam um tipo de objeto específico.
- Apesar de parecerem semelhantes às classes de linguagens de programação orientadas a objetos tradicionais, como Java ou C++, as classes em JavaScript são, na verdade, **açúcar sintático** por cima do sistema de protótipos já existente na linguagem.

Açúcar Sintático

- Em ciência da computação, um açúcar sintático é uma sintaxe dentro da linguagem de programação que tem por finalidade tornar suas construções mais fáceis de serem lidas e expressas.
- Ela faz com que o uso da linguagem se torne "mais doce" para o uso humano, permitindo que suas funcionalidades sejam expressas mais claramente, mais concisamente ou, ainda, como um estilo alternativo preferido por alguns.



Funcionamento das Classes em JavaScript

Sintaxe mais familiar

- As classes em JavaScript oferecem uma sintaxe **mais familiar** para definir objetos e suas propriedades, permitindo que desenvolvedores que vêm de linguagens orientadas a objetos se sintam mais confortáveis.

Métodos e Construtores

- Dentro de uma classe, você pode **definir métodos**, incluindo um método especial chamado **constructor**, que é invocado quando um objeto é instanciado a partir da classe. O **constructor** é usado para inicializar as propriedades de um objeto.

Herança

- As classes em JavaScript também permitem a criação de hierarquias de herança, usando a palavra-chave **extends** para herdar propriedades e métodos de outra classe.

Exemplo

```
class Animal {  
  constructor(tipo) {  
    this.tipo = tipo;  
  }  
  
  emitirSom() {  
    console.log(`O ${this.tipo} faz um som.`);  
  }  
}
```

// Classe Gato herda de Animal

```
class Gato extends Animal {  
  constructor(nome) {  
    super('gato');  
    this.nome = nome;  
  }  
  
  miar() {  
    console.log(`${this.nome} está miando.`);  
  }  
}
```

// Criando uma instância da classe Gato

```
const meuGato = new Gato('Bolinha');
```

```
meuGato.emitirSom(); // Saída: O gato faz um som.
```

```
meuGato.miar(); // Saída: Bolinha está miando.
```

No exemplo...

- **Animal** é uma classe base que define um método **emitirSom**, enquanto a classe **Gato** herda as propriedades e métodos de **Animal** usando a palavra-chave **extends**.
- A classe **Gato** também possui um método **miar** próprio.

Por fim...

- Embora as classes em JavaScript pareçam fornecer um modelo de programação orientada a objetos mais tradicional, é importante notar que, por baixo dos panos, elas continuam a se basear no sistema de **protótipos do JavaScript**.
- As classes em JavaScript oferecem uma **camada de abstração** mais amigável para o desenvolvedor e podem ser úteis para organizar o código de forma mais clara e fácil de entender, especialmente para aqueles familiarizados com outros paradigmas orientados a objetos.

Referências

- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Working_with_Objects

DESIGN DE INTERFACE PARA WEB

AULA 8 – UTILIZANDO OBJETOS EM JAVASCRIPT

WALTER TRAVASSOS SARINHO

@WALTEROPROFESSOR

WALTER.TRAVASSOS@UNIPE.EDU.BR

