INSTITUTO FEDERAL DE MATO GROSSO DO SUL
CAMPUS NAVIRAÍ

NOTAS DE AULA DE

LINGUAGEM DE APRESENTAÇÃO E ESTRUTURAÇÃO DE CONTEÚDOS II

Prof. MSc. Luiz F. Picolo

NAVIRAÍ - MS

1 Introdução

Aprender é a única coisa que a mente nunca se cansa,
nunca tem medo e nunca se arrepende
Autor desconhecido

1.1 O que é JavaScript

O JavaScript foi criado na década de 90 por **Brendan Eich** a serviço da Netscape (uma empresa de serviços de computadores nos EUA a qual era mais conhecida pelo seu navegador, o Netscape). Essa década foi um período de "revolução", pois os navegadores ainda eram estáticos sendo o mais popular dessa época o Mosaic, da NCSA.

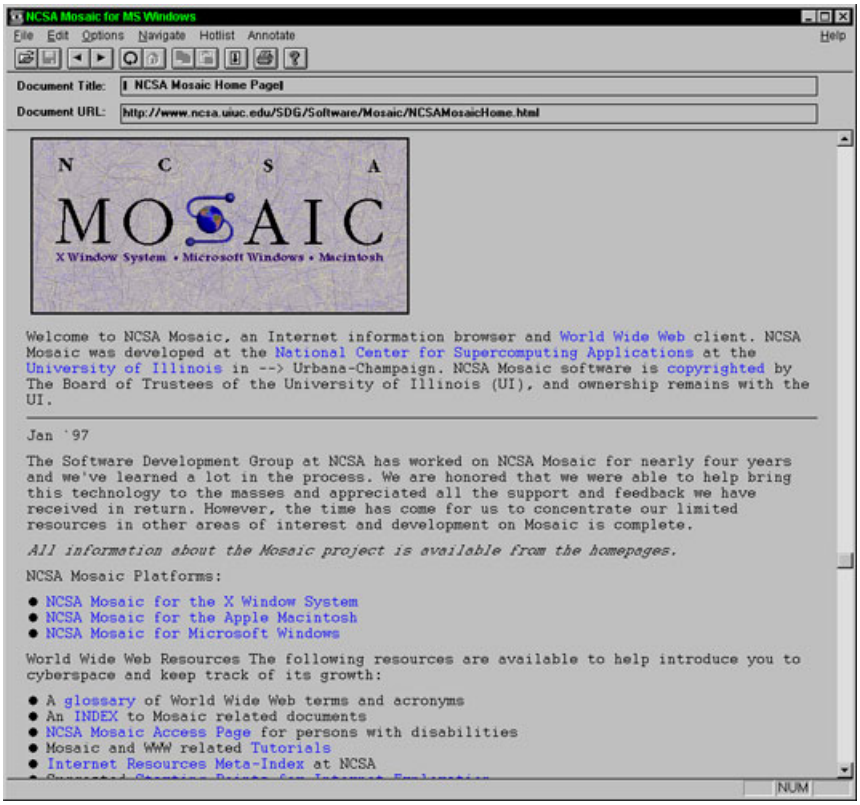

Figura 1 – NCSA Mosaic versão beta

Fonte: [History Computer]

O JavaScript foi introduzido em 1995 como uma forma de adicionar programas às páginas web do navegador Netscape. A linguagem, desde então, foi adotada por todos os outros grandes navegadores web que possuem interfaces gráficas. Ele tornou as aplicações

modernas possíveis, fazendo com que você não tenha que recarregar a página inteira quando for necessário realizar interações diretas com a aplicação. Além disso, ele é usado em páginas web mais tradicionais, fornecendo diferentes maneiras de criar interatividade e inteligência [Haverbeke 2014].

1.2 ECMAScript ou JavaScript?

Depois que o JavaScript foi adotado fora do Netscape, um documento padrão foi escrito para descrever a maneira na qual a linguagem deveria funcionar, garantindo que as diferentes partes dos softwares que afirmavam suportar JavaScript estavam, de fato, falando sobre a mesma linguagem. Esse documento é chamado de padrão ECMAScript, nomeado pela organização internacional Ecma, que foi responsável pela padronização. Na prática, os termos ECMAScript e JavaScript podem ser usados como sinônimos, pois são dois nomes para a mesma linguagem.

Na prática, existem diversos softwares que suportam JavaScript e possuem seu comportamente semelhante. Os navegadores, ou browsers, são exemplos destes tipos de software os quais implementam a linguagem por meio das especificações regidas pelo ECMAScript. Outro mais atual é o NodeJS ou simplesmente Node (https://nodejs.org/) que busca executar o JavaScript diretamente no servidor (Node será aprofundado em capítulos posteriores).

1.3 Executando JavaScript

Para constatar o fato citado na sessão anterior, vamos executar o seguinte código no console em diferentes navegadores. Para tanto, crie um arquivo **index.html** e um outro com o nome de **javascript.js**. No arquivo index.html adicione o seguinte código:

```
1  <!DOCTYPE HTML>
2  <html lang="pt-br">
3  <head>
4      <meta charset="utf-8">
5      <title>JS Exemplos</title>
6  </head>
7  <body>
8    <script type="text/javascript" src="javascript.js"></script>
9  </body>
10 </html>
```

E no arquivo **javascript.js** adicione o seguinte:

```
1 alert('Bem vindo(a) ao JavaScript')
```

O código acima possui o mesmo comportamento? Sim, o comportamento é o mesmo em todos os navegadores. Contudo, a forma gráfica com que é apresentado faz parte da implementação feita. Portanto, o desenvolvedor pode utilizar o JavaScript nos navegadores sem muito problema, pois eles seguem não ideias da equipe que o criou mas uma especificação que dita as regras de como determinadas funções devem se comportar.

1.4 Versões do JavaScript ou edições ECMAScript

Como foi visto, o ECMAScript é apenas a especificação. Contudo, ao estudar a linguagem é muito comum escutar a abreviação de ECMAScript, ou seja, ES. Assim, sempre que se ler ES seguido de um número, esse está fazendo referência a uma edição do ECMAScript. Atualmente, existem oito edições do ECMAScript publicadas, sendo que, a partir de 2015, as edições começaram a receber o ano e não mais o número da edição.

1. ECMAScript 1 (1997)
2. ECMAScript 2 (1998)
3. ECMAScript 3 (1999)
4. ECMAScript 4 Nunca foi lançada.
5. ECMAScript 5 (2009)
    5.1. ECMAScript 5.1 (2011)
6. ECMAScript 2015
7. ECMAScript 2016
8. ECMAScript 2017
9. ECMAScript 2018

1.5 Conclusão

Portanto, JavaScript se tornou a linguagem de programação mais popular no desenvolvimento Web sendo suportada por todos os navegadores e responsável por praticamente qualquer tipo de dinamismos em páginas web. Ao se usar todo o poder que ela tem para oferecer, pode-se chegar a resultados impressionantes. Alguns excelentes exemplos disso são aplicações Web complexas como Gmail, Google Maps e Google Docs.

2 Tipos de dados, Variáveis e Constantes

Aquele que não luta pelo futuro que quer,
deve aceitar o futuro que vier
Autor desconhecido

Definir o que são variáveis ou constantes de forma geral é extremamente simples. Uma variável nada mais é do que um espaço que o sistemas operacional (SO) aloca na memória de seu computador para que seja armazenado algo. Ou seja, quando precisamos armazenar um determinado valor nosso SO cria um espaço e permite ao usuário determinar o que será adicionado. A medida que não será utilizado este espaço com um valor, ele será dissipado, fazendo com que seus disponibilizando-os para serem usados na construção de outros novos valores [Haverbeke 2014].

Mas como é criado este valores? Eu preciso entender de memoria, sistemas operacionais, entre outros para poder alocar este valores? A resposta é não, não é necessário a compreenção de tão baixo nível, basta que se compreenda como é criar variáveis ou constantes em um determinada linguagem, que, neste caso, é JavaScript.

2.1 Tipos de dados

Diferente das linguagens com tipagem forte, ou seja, que levam em consideração o tipo para executar suas operações, o JavaScript possui tipagem fraca, a qual é dado o tipo dos dados que serão alocados quando o código é executado. Outro ponto é a tipagem dinâmica. Logo, não é necessário tipar uma variável quando inicia-se o programa como é feito em Java, C, C++, entre outras linguagens, mas esta responsabilidade é repassada ao compilador, que, caso do JavaScript em questão, esse processo é dado o nome Inferência de Tipo.

A inferência de tipos é a capacidade do compilador entender/'adivinhar' qual é o tipo de dados de determinada variável sem ela ter sido declarada no código escrito.

Segundo a MDN Web Docs Mozilla 2019, JavaScript reconhece os seguintes tipos de valores:

- Números (en), como 42 ou 3,14159
- Valores lógicos (Booleanos) (en), true ou false

- Strings (en), tais como "Howdy!"
- null, um palavra chave especial denotando um valor nulo; null também é um valor primitivo. Como JavaScript é sensível a maiúsculas, null não é a mesma coisa que Null, NULL, ou qualquer outra variante
- undefined (en), uma propriedade de alto nível a qual possui o valor indefinido; undefined também é um valor primitivo.

Este conjunto de tipos de valores, relativamente pequeno, permite realizar funções úteis nas aplicações. Não há distinção explícita entre números inteiros e reais, para JavaScript, todos os números são tratados como *Number*.

2.2 Variáveis

O JavaScript é uma linguagem de tipagem dinâmica. Isto significa que não é necessário especificar o tipo de dado de uma variável quando ela for declarada, e tipos de dados são automaticamento convertidos conforme necessário durante a execução do script. Então, por exemplo, pode-se definir uma variável como:

```
1  var idade = 30
2  var instituicao = "IFMS"
```

Para que possamos saber o tipo que foi atribuido para cada variável, podemos utilizar a função **typeof** da seguinte forma:

```
1  var idade = 30
2  console.log(typeof idade) // Retornará number
```

Outra forma de declarar variáveis é utilizando a palavra reservada **let**.

```
1  let idade = 30
2  let instituicao = "IFMS"
```

Mas qual a diferença entre **var** e **let**? Neste momento não possuimos o conhecimento necessário para diferenciar as duas palavras reservadas. Contudo, veremos nos próximos capítulos como as duas se diferenciam e quando utilizar uma ou outra.

2.2.1 Hoinsting

Segundo a MDN Web Docs Mozilla 2019, em JavaScript, funções (a qual será vista em um outro capítulo) e variáveis são hoisted (ou "levados ao topo"). Hoisting é um

comportamento do JavaScript de mover declarações para o topo de um escopo (o escopo global ou da função em que se encontra). Isso significa que nós somos capazes de usar uma função ou variável antes mesmo de tê-las declaradas, ou em outras palavras: uma função ou variável podem ser declaradas depois de já terem sido utilizadas.

```
1  foo = 2
2  var foo;
3
4  // é implicitamente entendido como:
5  var foo;
6  foo = 2;
```

Assim, as variáveis quando são declaradas sem um valor, recebem o valor **undefined**, ou seja, sem tipo.

2.3 Constantes

Podemos criar elementos "somente leitura", nomeados constantes com a palavra chave **const**. A sintaxe de um identificador constante é a mesma para um identificador de variáveis: deve começar com uma letra ou sublinhado e pode conter caracteres alfabéticos, numéricos ou sublinhado. Porém, uma constante não pode ter seu valor mudado por meio de uma atribuição ou ser declarada novamente enquanto o *script* estiver rodando.

```
1  const idade = 30
2  console.log(typeof idade) // Retornará number
```

2.4 Tipo básico de dados

Essa sessão introduz os elementos que representam os tipos de valores e os operadores que podem atuar sobre eles presentes na linguagem que estamos estudanto.

2.4.1 Números

Em JavaScript todos números são tratados como *Number*. Ao criar uma variável contendo um número inteiro ou decimal qualquer, esta variável em algumas linguagem seriam tratadas como Integer ou Float por exemplo. Contudo,em JavaScript, esta recebe o tipo *Number* independente do seu valor.

```
1  var idade = 30
2  console.log(typeof idade) // Retornará number
```

```
3
4   var peso = 65.5
5   console.log(typeof peso) // Retornará number
```

2.4.1.1 Números Especiais

Existem três valores especiais no JavaScript que são considerados números, mas não se comportam como números normais.

Os dois primeiros são Infinity e -Infinity, que são usados para representar os infinitos positivo e negativo. O cálculo Infinity - 1 continua sendo Infinity, assim como qualquer outra variação dessa conta. Entretanto, não confie muito em cálculos baseados no valor infinito, pois esse valor não é matematicamente sólido e rapidamente nos levará ao próximo número especial: NaN [Haverbeke 2014].

NaN é a abreviação de "not a number" (não é um número), mesmo sabendo que ele é um valor do tipo número. Você receberá esse valor como resultado quando, por exemplo, tentar calcular 0 / 0 (zero dividido por zero), Infinity - Infinity ou, então, realizar quaisquer outras operações numéricas que não resultem em um número preciso e significativo [Haverbeke 2014].

2.4.2 Strings

O próximo tipo básico de dado é a String. Strings são usadas para representar texto, e são escritas delimitando o seu conteúdo entre aspas. Ambas as aspas simples e duplas podem ser usadas para representar Strings, contanto que as aspas abertas sejam iguais no início e no fim.

```
1   var nome = "Genoveva"
2   console.log(typeof nome) // Retornará String
```

Existem outras maneiras de manipular as Strings, as quais serão discutidas nos próximos capitulos

2.4.3 Valores Booleanos

Você frequentemente precisará de um valor para distinguir entre duas possibilidades, como por exemplo "sim" e "não", ou "ligado" e "desligado". Para isso, o JavaScript possui o tipo Booleano, que tem apenas dois valores: verdadeiro e falso (que são escritos como true e false respectivamente).

```
1   console.log(3 > 2) // Retornará true
```

```
2  console.log(3 < 2) // Retornará false
```

2.4.4 Valores Indefinidos

Existem dois valores especiais, null e undefined, que são usados para indicar a ausência de um valor com significado. Eles são valores por si sós, mas não carregam nenhum tipo de informação. A diferença de significado entre undefined e null é um acidente que foi criado no design do JavaScript, e não faz muita diferença na maioria das vezes. Podemos comprovar isso comparando ambos

```
1  console.log(null == undefined); // Retornará True
```

2.5 Aritmética

Para que possamos realizar os cálculos básicos em JavaScript, podemos usar os operadores que já estamos acostumados

```
1  2 + 2 // Soma
2  5 - 4 // Subtrai
3  2 * 3 // Multiplica
4  4 / 2 // Divide
5  144 % 12 // Retorna o Resto da divisão
```

Mas o que acontece quando adicionamos os operados acima a Strings?

```
1  "Java" + "Script" // JavaScript
2  "Java" - "Script" // NaN
3  "Java" * "Script" // NaN
4  "Java" / "Script" // NaN
5  "Java" % "Script" // NaN
```

Ou seja, com o valor Aritmético + as Strings são concatenadas.

2.6 Exercícios de fixação

1. Crie três variáveis contendo três notas. Depois, calcule a média das três notas e mostre-a na tela por meio de um **Alert**.

2. Crie cinco variáveis contendo cinco números inteiros. Logo após imprima o quadrado de cada número.

2.7 Concatenação e interpolação de Strings

Concatenar uma String nada mais é do que executar a união entre duas partes. Como visto, podemos concatenar duas string utilizando o operador +. da seguinte forma:

```
1 var string1 = "Java";
2 var string2 = "Script";
3
4 console.log(string1 + string2)
```

Mas, existe uma nova fora para que possamos unir strings, que usa a ideia de interpolação.

```
1   var string1 = "Java";
2   var string2 = "Script";
3
4   \\ Usaremos o simbolo da crase
5   console.log(`${string1}${String2}`)
```

2.8 prompt e confirm

O ambiente fornecido pelos navegadores contém algumas outras funções para mostrar janelas. Você pode perguntar a um usuário uma questão Ok/Cancel usando confirm. Isto retorna um valor booleano: true se o usuário clica em OK e false se o usuário clica em Cancel [Haverbeke 2014].

```
1   confirm("Deseja␣realmente␣deletar␣este␣dado?")
```

Já o comando **prompt** pode ser usado para criar uma questão "aberta". O primeiro argumento é a questão; o segundo é o texto que o usuário inicia. Uma linha do texto pode ser escrita dentro da janela de diálogo, e a função vai retornar isso como uma string.

```
1 prompt("Digite␣seu␣nome")
```

2.9 Exercicios de fixação

1. Faça um programa que leia duas string e exiba uma mensagem com o resultado da concatenação das mesmas.

2. Faça um programa que leia dois números. Logo após, some os mesmos e exiba uma mensagem utilizando o console.log com o resultado. Pesquisa a forma como podemos converter String em Number em JavaScript

2.10 Condicional

Quando seu programa contém mais que uma declaração, as declarações são executadas, previsivelmente, de cima para baixo. Assim, caso tenhamos duas condições, precisamos de algum mecanismo que possa nos auxiliar a executar uma condição ou outra. Assim, tanto no JavaScript quanto em outras linguagens de programação, podemos utilizar a palavra-chave reservada **if** para expressar uma condição. No caso mais simples, nós queremos que algum código seja executado se, e somente se, uma certa condição existir.

```
1  var idade = Number(prompt('Digite sua idade'));
2  if (idade >== 18){
3    alert("Maior de idade")
4  }
```

Contudo, normalmente teremos que executar uma condição se algo for **verdadeiro** e outra caso o resultado seja **falso**. Assim, esse caminho alternativo é representado pela palavra reservada **else**.

```
1  var idade = Number(prompt('Digite sua idade'));
2  if (idade <== 18){
3    alert('Maior de idade')
4  } else {
5    alert('Menor de idade')
6  }
```

Se tivermos mais que dois caminhos a escolher, múltiplos pares de **if**/**else** podem ser "encadeados".

```
1  var idade = Number(prompt('Digite sua idade'));
2  if (idade === 18){
3    alert('Tem 18 anos')
4  } else if (idade < 18){
5    alert('Menos de 18 anos)
6  } else {
7    alert('Maior de 18 anos)
8  }
```

2.11 Exercício de fixação

1. Faça um programa que peça duas notas de um estudante. Em seguida calcule a média do aluno e apresente o resultado, Reprovado ou Aprovado. Lembre-se, a média deve ser igual ou maior a 7,0.

2. Faça um programa que leia três números e mostre-os, no console, os mesmos em ordem decrescente.

2.12 laços de repetição

Frequentemente em nossas aplicações precisamos repetir a execução de um bloco de códigos do programa até que determinada condição seja verdadeira, ou senão até uma quantidade de vezes seja satisfeita. Para que essas repetições sejam possíveis, usamos os laços de repetições.

2.12.1 Loops While e Do

Uma declaração que inicia com a palavra-chave while cria um loop. A palavra while é acompanhada por uma expressão entre parênteses e seguida por uma declaração, similar ao if. O loop continua executando a declaração enquanto a expressão produzir um valor que, após convertido para o tipo Booleano, seja true [Haverbeke 2014].

```
1  var numero = 0;
2  while (numero <= 12) {
3    console.log(numero);
4    numero = numero + 2;
5  }
```

Já o loop do é uma estrutura de controle similar ao while. A única diferença entre eles é que o do sempre executa suas declarações ao menos uma vez e inicia o teste para verificar se deve parar ou não apenas após a primeira execução. Para demonstrar isso, o teste aparece após o corpo do loop:

```
1  do {
2    var name = prompt("Digite␣seu␣nome");
3  } while (!name);
4  console.log(name);
```

Esse programa irá forçar você a informar um nome. Ele continuará pedindo até que seja fornecido um valor que não seja uma *string* vazia.

2.12.2 Loops For

O JavaScript, assim como outras linguagens de programação, fornece uma forma um pouco mais curta e compreensiva para que possamos realizar a mesma função do Loop **While** chamada de loop **for**. Neste Loop, a execução acontece até o momento em que a segunda alternativa, que neste caso é **number** $<=$ **12** seja verdadeira

```
1  for (var number = 0; number <= 12; number = number + 2){
2    console.log(number);
3  }
```

Contudo, ter uma condição que produza um resultado false não é a única maneira que um loop pode parar. Existe uma declaração especial chamada **break** que tem o efeito de parar a execução e sair do loop em questão.

```
1  for (var current = 20; ; current++) {
2    if (current % 7 == 0)
3    break;
4  }
5  console.log(current);
```

2.13 SwitchCase

É comum que, com o aumento da complexidade dos algorítimos desenvolvidos, o código fique assim:

```
1  if (variable == "value1") action1();
2  else if (variable == "value2") action2();
3  else if (variable == "value3") action3();
4  else defaultAction();
```

Há um construtor chamado switch que se destina a resolver o envio de valores de uma forma mais direta. Infelizmente, a sintaxe JavaScript usada para isso (que foi herdada na mesma linha de linguagens de programação, C e Java) é um pouco estranha - frequentemente uma cadeia de declarações if continua parecendo melhor. Aqui está um exemplo:

```
1  switch (prompt("What␣is␣the␣weather␣like?")) {
2  case "rainy":
3    console.log("Remember␣to␣bring␣an␣umbrella.");
4    break;
```

```
5   case "sunny":
6     console.log("Dress lightly.");
7     break;
8   case "cloudy":
9     console.log("Go outside.");
10    break;
11  default:
12    console.log("Unknown weather type!");
13    break;
14  }
```

2.14 Exercícios de fixação

1 - Escreva um programa que faça sete chamadas a console.log() para retornar o seguinte triângulo:

```
1  #
2  ##
3  ###
4  ####
5  #####
6  ######
7  #######
```

2 - Escreva um programa que cria uma string que representa uma grade 8x8, usando novas linhas para separar os caracteres. A cada posição da grade existe um espaço ou um caractere. Esses caracteres formam um tabuleiro de xadrez.

Passando esta string para o console.log deve mostrar algo como isto:

```
1  # # # #
2   # # # #
3  # # # #
4   # # # #
5  # # # #
6   # # # #
7  # # # #
8   # # # #
```

3 - Transforme o código abaixo utilizando a sintaxe do **SwitchCase**.

```
1  var ano = 6;
```

```
2
3   if (anos <= 1) {
4     console.log("Iniciante");
5   } else if (anos <= 3) {
6     console.log("Intermediário");
7   } else if (anos <= 6) {
8     console.log("Avancado");
9   } else {
10    console.log("Jedi Master");
11  }
```

3 Funções

Todo mestre já foi aprendiz um dia.
Mas também engana-se quem pensa que a busca por
conhecimento tem um fim.
Autor desconhecido

Nós, até este momento, já observamos valores de funções. O **alert**, **prompt**, **confirm** são funções que podemos chamar em qualquer parte do nosso código, mas não sabemos como ela realiza determinado processo para chegar a determinados resultados. Assim, podemos afirmar portanto que o conceito de encapsular uma parte do programa em um valor tem vários usos. É uma ferramenta usada para estruturar aplicações de larga escala, reduzir repetição de código, associar nomes a subprogramas e isolar esses subprogramas uns dos outros.

3.1 Definindo uma Função

A definição da função (também chamada de declaração de função) consiste no uso da palavra chave *function*, seguida por:

1. Nome da Função.
2. Lista de argumentos para a função, entre parênteses e separados por vírgulas.
3. Declarações JavaScript que definem a função, entre chaves .

Por exemplo, o código a seguir define uma função simples chamada elevarAoQuadrado:

```
1  function elevarAoQuadrado(numero) {
2    return numero * numero;
3  }
```

A função elevarAoQuadrado recebe um argumento chamado numero. A função consiste em uma instrução que indica para retornar o argumento da função (isto é, numero) multiplicado por si mesmo. A declaração **return** especifica o valor retornado pela função.

```
1  return numero * numero;
```

3.2 Chamando funções

A definição de uma função não a executa. Definir a função é simplesmente nomear a função e especificar o que fazer quando a função é chamada. Chamar a função executa realmente as ações especificadas com os parâmetros indicados. Por exemplo, se você definir a função elevarAoQuadrado, você pode chamá-la do seguinte modo:

```
1  elevarAoQuadrado(5);
```

Para apresentar os dados podemos utilizar o console.log da seguinte forma:

```
1  console.log(elevarAoQuadrado(5))
```

Lembre-se: funções não devem apresentar valores mas apenas retorná-los.

3.3 Expressão de função

Embora a declaração de função acima seja sintaticamente uma declaração, funções também podem ser criadas por uma expressão de função. Tal função pode ser anônima; ele não tem que ter um nome. Por exemplo, a função elevarAoQuadrado poderia ter sido definida como:

```
1  var elevarAoQuadrado = function(numero) {return numero * numero};
2  var x = elevarAoQuadrado(4) //x recebe o valor 16
```

Ou podemos também utilizar as **Arrow Functions**. Neste caso, removeremos a palavra reservada *function* e adicionaremos a Arrow ou Fecha/Seta. (Todas as expressões buscam resolver determinados problemas. Contudo, não vamos abardá-los neste texto)

```
1 var elevarAoQuadrado = (numero) => {return numero * numero};
2 var x = elevarAoQuadrado(4) //x recebe o valor 16
```

3.4 Exercícios de Fixação

1. Escreva uma função que recebe dois argumentos e retorna o menor deles. A função deve ser invocada da seguinte forma:

```
1  minimo(2, 4) // Deverá retornar 2
```

2. Vamos criar uma função para um duelo. Crie uma função que receba o nome de dois personagens e que sera invocada da seguinte forma:

```
1   dueloRPG("Mago", "Guerreiro") // Poderá retornar da seguinte forma
       'Guerreiro ganhou com dano de: 0.12312'
```

Para tanto, use a função **Math.random()** para gerar um número aleatório que será o dano de seu personagem.

4 O Modelo de Objeto de Documentos (DOM)

Todo mestre já foi aprendiz um dia.
Mas também engana-se quem pensa que a busca por
conhecimento tem um fim.
Autor desconhecido

Você pode imaginar um documento HTML como um conjunto de caixas aninhadas. Tags como e encapsulam outras tags, as quais, por sua vez, contêm outras tags ou texto. Aqui está o documento de exemplo do último capítulo:

```
<html>
  <head>
    <title>Minha home page</title>
  </head>
  <body>
    <h1>Minha home page</h1>
    <p>Olá, eu sou Marijn e essa é minha home page.</p>
    <p>Eu também escrevi um livro! leia-o
    <a href="http://eloquentjavascript.net">aqui</a>.</p>
  </body>
</html>
```

Essa página tem a seguinte estrutura:

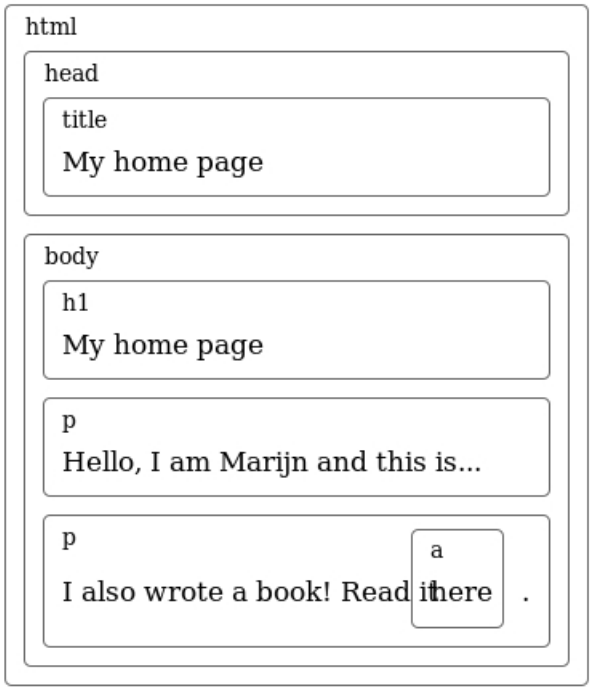

Figura 2 – Containers HTML

Fonte: [Haverbeke 2014]

A estrutura de dados que o navegador usa para representar o documento segue este formato. Para cada caixa há um objeto, com o qual podemos interagir para descobrir coisas como: qual tag HTML ele representa e quais caixas e textos ele contém. Essa representação é chamada de Modelo de Objeto de Documentos, também apelidada de DOM (do inglês Document Object Model).

A variável global document nos dá acesso à esses objetos. Sua propriedade documentElement se refere ao objeto que representa a tag . Essa propriedade também nos fornece as propriedades head e body, alocando objetos para esses elementos.

Contudo, ao analisarmos a estrutura do HTML, podemos perceber que alguns elementos estão contidos dentro de outros, e assim, sucessivamente. Esse conjunto de elementos pode ser representado em formato de **árvore**, também conhecida como árvore DOM.

O DOM (Document Object Model), segundo Leonardo 2018 é uma interface que representa como os documentos HTML e XML são lidos pelo seu browser. Após o browser ler seu documento HTML, ele cria um objeto que faz uma representação estruturada do seu documento e define meios de como essa estrutura pode ser acessada. Nós podemos acessar e manipular o DOM com JavaScript, é a forma mais fácil e usada.

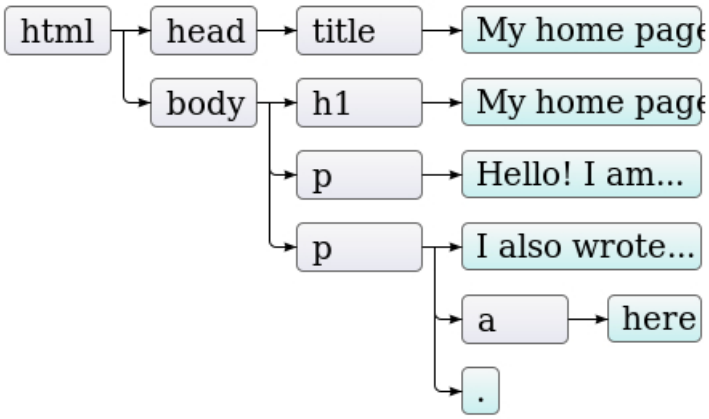

Figura 3 – Árvore DOM

Fonte: [Haverbeke 2014]

4.1 Manipulando o DOM

O DOM possui muitos métodos, são eles que fazem a ligação entre os nodes (elementos) e os eventos. O primeiro que veremos e o **write** e o **writeln**.

Em nossas aulas, quando desejávamos mostrar um resultando, usávamos o console.log ou o alert. Contudo, por meio da manipulação do DOM, podemos printar esses dados diretamente na tela.

```
1  document.write("Hello World")
2  document.writeln("Hello World")
```

Por meio dos comandos acima, todo o dom e limpo e apenas a palavra Hello World é printada na tela.

4.1.1 getElementById()

Esse método retorna o elemento que estiver contendo o nome do ID passado. Como os IDs devem ser únicos, é um método muito útil para pegar apenas o elemento desejado.

```
1   var idTag = document.getElementsById('idTag');
```

Exemplo

Neste exemplo veremos o evento OnClick. Não se preocupe, este será mais aprofundado nos próximos capítulos.

```
1  <html>
```

```
2     <head>
3       <title>getElementById example</title>
4     </head>
5     <body>
6       <p id="para">Some text here</p>
7       <button onclick="MudarCor('blue');">blue</button>
8       <button onclick="MudarCor('red');">red</button>
9     </body>
10  </html>
```

```
1   function MudarCor(cor) {
2     var elem = document.getElementById('para');
3     elem.style.color = cor;
4   }
```

4.1.2 getElementsByClassName()

Esse método retorna um **HTMLCollection** de todos elementos que estiverem contendo o nome da classe passada.

```
1   var x = document.getElementsByClassName("exemplo");
```

```
1   <!DOCTYPE html>
2   <html>
3     <body>
4       <div class="exemplo">Primeiro nome</div>
5       <div class="exemplo">Segundo nome</div>
6       <button onclick="mudarTexto()">Try it</button>
7       <script>
8         function mudarTexto() {
9           var x = document.getElementsByClassName("exemplo");
10          x[0].innerHTML = "Novo texto";
11        }
12      </script>
13    </body>
14  </html>
```

Referências

HAVERBEKE, M. *Eloquent javascript: A modern introduction to programming*. [S.l.]: No Starch Press, 2014. Citado 7 vezes nas páginas 3, 5, 8, 10, 12, 20 e 21.

HISTORY Computer. Disponível em: <https://history-computer.com/Internet/Conquering/Mosaic.html>. Acesso em: 09 julho. 2019. Citado na página 2.

LEONARDO, M. *Entendendo o DOM (Document Object Model)*. 2018. Disponível em: <https://tableless.com.br/entendendo-o-dom-document-object-model/>. Citado na página 20.

MDN Web Docs Mozilla. 2019. Disponível em: <https://developer.mozilla.org/pt-PT/docs/Web/JavaScript/Guia/Valores,_Variáveis_e_Literais>. Citado 2 vezes nas páginas 5 e 6.