Javascript [1]

Um Breve Tutorial

Gabriel P. Silva[2]

14 de Agosto de 2018

[2]gabriel@dcc.ufrj.br

Dedicado aos meus filhos, Vinícius, Danilo, Caio Vítor e a Madalena, minha constante inspiração.

Conteúdo

1 O que é Javascript? 3

2 Características 5

2.1 Introdução 5

2.2 HTML DOM 5

2.3 Entrada e saída 7

2.3.1 No navegador 7

2.3.1.1 Na console do node.js ou navegador 8

2.4 Inserindo Código Javacript 9

2.5 Escrevendo no Documento HTML 10

2.6 Variáveis e Comentários 12

2.6.1 Variáveis 12

2.6.2 Comentários 12

3 Operadores e Tipos de Dados 13

3.1 Operadores 13

3.2 Tipos de Dados 14

3.2.1 Arrays 14

3.2.2 Métodos Pré-definidos para Arrays 15

3.3 Conversão de Tipos 16

3.3.1 Conversão Implícita de Tipos 16

3.3.2 Conversão de Explícita de Tipos 17

4 Sentenças Condicionais e Laços 18

4.1 if-then-else 18

4.2 Laços com while e for 18

4.3 Switch 19

5 Eventos 20

5.1 Elementos HTML Usuais 21

6 Objetos 23

7 Funções **25**
7.1 Argumentos de Funções 26
7.2 Operador Spread 26
7.3 Funções Anônimas 27
7.4 Recursividade em Funções 27
7.5 Funções anexadas a Objetos 27
7.5.1 Protótipos de Funções 29

8 Node.js **30**
8.1 npm 31
8.2 Gerenciando a Cache 33

9 JSON **35**

Lista de Figuras

2.1 DOM HTML 6

9.1 object 35
9.2 array 36
9.3 value 36
9.4 string 37
9.5 number 37

Lista de Tabelas

3.1 Operadores Aritméticos 14
3.2 Tipos de Dados 14

Prefácio

Este é um tutorial muito simples com o objetivo de apresentar os conceitos básicos da linguagem Javascript. É uma obra que pretendemos tenha um caráter dinâmico, evoluindo e sendo aprimorada ao longo do tempo.

O JavaScript é uma linguagem de programação que permite implementar funções complexas em páginas da web. Toda vez que uma página faz mais do que apenas exibir informações estáticas - exibindo periodicamente atualizações de conteúdo, mapas interativos ou gráficos 2D/3D animados, ou ainda rolando caixas de escolha de vídeo, e assim por diante - você pode apostar que o JavaScript provavelmente está envolvido.

Ao contrário do que alguns possam imaginar, a linguagem Javascript tem pouca ou nenhuma relação com a linguagem Java. A linguagem Javascript é interpretada em tempo de execução, ou seja, ao contrário do Java, não é gerado nenhum código objeto, nem mesmo para uma máquina virtual, no processo de sua execução.

Algumas características marcantes do Javascript são: é uma linguagem orientada a objeto, onde não há nenhuma distinção entre tipos de objetos. A herança é feita através do mecanismo de protótipo, e as propriedades e métodos podem ser adicionados a qualquer objeto dinamicamente. Os tipos de dados das variáveis não são declarados (tipagem dinâmica, tipagem fraca).

É uma linguagem que tem evoluído bastante em diversos aspectos, tendo uma aceitação e utilização cada vez maior, ao longo dos anos. Apesar de ser uma linguagem interpretada, o uso de técnicas empregadas na sua interpretação, como *just in time*, tem feito com que seu desempenho se aproxime bastante daquele obtido quando se faz uso de código nativo.

Neste tutorial apresentaremos os conceitos básicos da linguagem Javascript, além do uso do Node.js®, que é máquina de execução para o Javascript construída com base na versão 8 máquina de execução do navegador Chrome. O Node.js usa um modelo de E/S não bloqueante, dirigido a eventos, o que o torna leve e eficiente. O ecossistema de pacotes do Node.js, npm, que é o maior ecossistema de bibliotecas de código aberto no mundo, também será apresentado neste tutorial.

Finalmente, apresentamos brevemente o JSON (JavaScript Object Notation), que é um formato leve para a troca de dados, que é fácil para leitura e escrita por humanos, além de ser fácil para ser analisado e gerado por máquinas.

Agradecimentos

- Ao meus alunos, pela ajuda na criação desta obra com seus comentários e sugestões.

Gabriel P. Silva

http://gabrielsilva.rio.br/

1

O que é Javascript?

O JavaScript é uma linguagem de programação interpretada de alto nível, dinâmica e não-tipada. Quando aplicada a um documento HTML, pode fornecer interatividade dinâmica em sites. Foi inventada por Brendan Eich, quando trabalhava na Netscape, em 1995. Ele também é co-fundador do projeto Mozilla, da Fundação Mozilla e da Mozilla Corporation.

A linguagem foi padronizada na especificação ECMAScript, ECMA-262, tendo a sua primeira versão publicada em Junho de 1997. Desde então, diversas versões do padrão foram publicadas, sendo que o nome ECMAScript foi uma solução de compromisso entre as diversas empresas envolvidas na padronização da linguagem. A ultima versão da especificação foi publicada em Junho de 2016 [1].

Junto com o HTML e o CSS, o JavaScript é uma das três tecnologias principais para a produção de conteúdo para a World Wide Web: a maioria dos sites a utiliza e todo os navegadores modernos a suportam sem o uso de *plugins*.

O JavaScript é uma linguagem incrivelmente versátil, bastante compacta, mas muito flexível. Seus desenvolvedores escreveram uma grande variedade de ferramentas que complementam o núcleo da linguagem JavaScript, permitindo uma ampla quantidade de funcionalidade extra com o mínimo esforço. Essas incluem:

- Interfaces de Programação de Aplicativos (APIs) integradas em navegadores da Web, fornecendo funcionalidades como a criação dinâmica de HTML e configuração de estilos CSS; captura e manipulação de um fluxo de vídeo da webcam do usuário ou geração de gráficos 3D e amostras de áudio.

- APIs de terceiros para permitir que os desenvolvedores incorporem funcionalidade em seus sites de outros provedores de conteúdo, como o Twitter ou o Facebook.

- Estruturas e bibliotecas de terceiros que você pode aplicar ao seu HTML para permitir que você construa rapidamente sites e aplicativos.

[1] https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

Para se iniciar com o JavaScript é muito fácil: tudo que você precisa é um navegador moderno ou instalar o ambiente node.js.

Por exemplo, existem duas ferramentas incorporadas no Firefox que são úteis para experimentações com o JavaScript: a console *Web* e o *Scratchpad*.

A console *Web* mostra informações sobre a página atualmente carregada e também inclui uma linha de comando que pode ser usada para executar expressões de JavaScript.

Para abrir a console da Web (use a combinação de teclas Ctrl + Shift + I no Windows e Linux ou Cmd-Option-K no Mac), ou selecione "Web Console"no menu "Web Developer"do Firefox. A console aparece na parte inferior da janela do navegador. Ao longo da console existe uma linha de comando que você pode usar para inserir código JavaScript e a saída aparece no painel que fica imediatamente acima.

A console Web é ótima para executar uma única linha de JavaScript, mas para exemplos mais complexos, o *Scratchpad* é uma ferramenta melhor.

Para abrir o *Scratchpad* use a combinação de teclas (Shift + F4) ou selecione "Scratchpad"no menu "Developer", que está no menu no Firefox. Ele abre em uma janela separada e é um editor que você pode usar para escrever e executar JavaScript no navegador. Você também pode salvar código digitado no disco e carregá-los a partir do disco.

2

Características

2.1 Introdução

O JavaScript possui sintaxe intencionalmente similar ao Java (ou C++). A sintaxe do JavaScript é relaxada para permitir que seja utilizada como um linguagem de *scripting* fácil de ser usada. Por exemplo, uma variável não precisa ter seu tipo declarado, nem os tipos são associados com propriedades, e as funções definidas não precisam ter suas declarações aparecendo textualmente antes de uma chamada para elas. Com o Javascript podemos:

- Modificar conteúdo HTML
- Modificar atributos HTML
- Remover elementos e atributos HTML existentes
- Adicionar novos elementos e atributos HTML existentes
- Modificar estilos CSS
- Esconder elementos HTML
- Mostrar elementos HTML ocultos

2.2 HTML DOM

Quando uma página HTML é carregada, o navegador cria um Modelo Objeto de Documento (DOM), que é basicamente uma árvore de objetos da página, com a estrutura que pode ser vista na Figura 2.1.

Alguns objetos HTML importantes utilizados no Javascript são elencados a seguir:

- NAVIGATOR: O objeto navegador contém informações sobre o navegador utilizado.

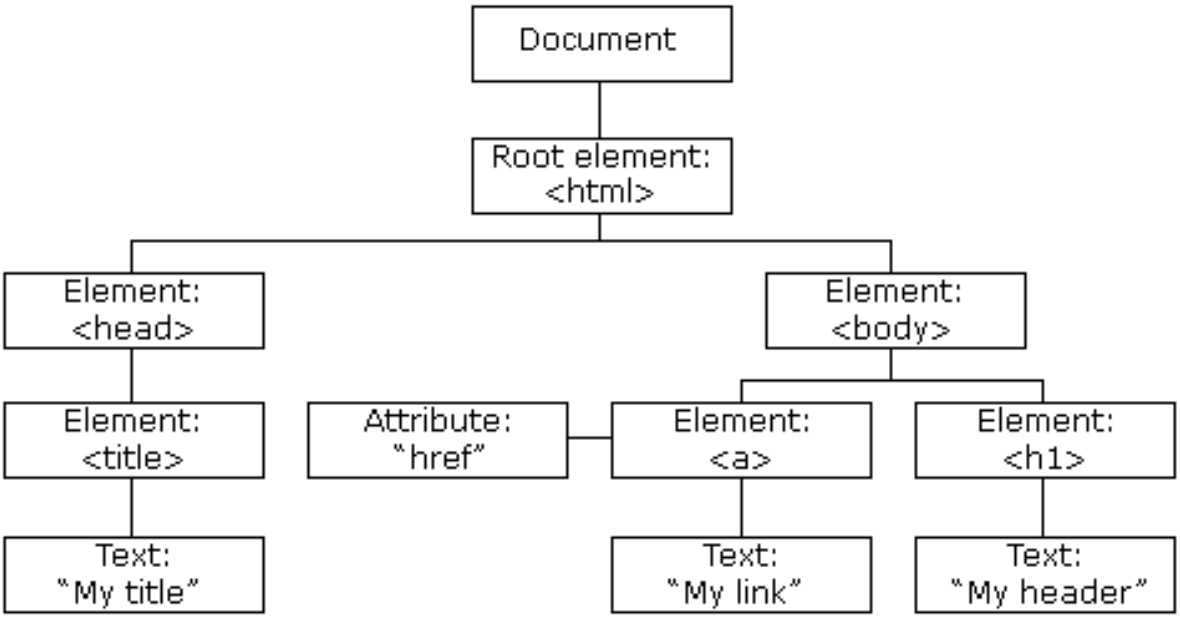

Figura 2.1: DOM HTML

- DOCUMENT: Quando um documento HMTL é carregado no navegador, ele se torna um objeto DOCUMENT. O objeto DOCUMENT é o nó raiz do documento HTML, é uma parte do objeto WINDOW e pode ser acessado como window.document.
- WINDOW: O objeto WINDOW representa uma janela aberta em um navegador. Se um documento contém "frames", o navegador cria um objeto WINDOW para o documento HTML e um objeto WINDOW adicional para cada "frame".
- CONSOLE: O objeto CONSOLE permite acesso à console de depuração do navegador.
- FORM: O objeto formulário do JavaScript é uma propriedade do objeto DOCUMENT, que corresponde a um formulário de entrada HTML construído com a etiqueta FORM. Um formulário pode ser enviado chamando o método de submissão (*submit()*) do JavaScript ou clicando no botão enviar do formulário.
- DATE: O objeto DATE permite armazenamento básico e recuperação de datas e horários.

Os objetos tem **propriedades** e **métodos** associados, sendo que as **propriedades** são características particulares dos objetos e os **métodos** são funções que podem ser aplicadas para criar, remover, modificar as propriedades associadas aos objetos e os próprios objetos em si.

Por exemplo, um objeto FORM representa um elemento <form> em HTML. Você pode acessar um elemento <form> com o uso do método getElementById():

```
var x = document.getElementById("myForm");
```

Você pode criar um elemento <form> usando o método document.createElement():

```
var x = document.createElement("FORM");
```

Alguns exemplos de propriedades de um elemento <form> são o *name, enctype, target*, etc. Exemplos de métodos aplicáveis a um elemento <form> são *reset()* (reinicia um formulário) e *submit()* (envia um formulário).

O Javascript permite ainda a captura e gerenciamento de eventos, tais como: *onClick, onSubmit, onMouseOver e onMouseOut*, que serão vistos em mais detalhes no Capítulo 5.

2.3 Entrada e saída

Uma das primeiras preocupações do programador é como realizar a entrada e saída, o que é uma questão particularmente interessante em um programa em Javascript, já que pode ser executado tanto pelo navegador como no ambiente node.js. A entrada e saída em Javascript pode ser realizada basicamente de duas formas:

2.3.1 No navegador

Podemos usar alguns métodos simples, como *prompt* ou *alert*. O primeiro faz a captura de um valor após escrever uma mensagem e o segundo apenas escreve uma mensagem em uma janela de "pop-up".

- **window.prompt**: Abre uma janela para pedir uma string ao usuário.

```
    nome = window.prompt("Digite o seu nome:", "");
```

- **window.alert**: Abre uma janela para exibir um aviso ao usuário.

```
    window.alert("Senha inválida.");
```

A seguir um exemplo mais completo, que utiliza algumas facilidades do Javascript, como a possibilidade de capturar eventos com o método *addEventListener()*. Observamos também a referência aos elementos HTML com o uso do método *getElementById()*.

```
<html>
<meta charset="UTF-8">
<head>
<title>Alô Mundo</title>
</head>
```

```
<body>

Nome: <input id="nome">
Sobrenome: <input id="sobrenome">
<button id="alo">Diga Alô!</button>

<hr>
<div id="result"></div>

<script>
function diga_alo() {
    var fname = document.getElementById('nome').value;
    var lname = document.getElementById('sobrenome').value;

    var html = 'Alô <b>' + fname + '</b> ' + lname;

    document.getElementById('result').innerHTML = html;
}

document.getElementById('alo').addEventListener('click', diga_alo);
</script>

</body>
</html>
```

2.3.1.1 Na console do node.js ou navegador

A entrada e saída na console se dá de uma maneira totalmente diferente. A seguir um exemplo de como imprimir mensagens de uma forma bem simples na console.

- **console.log**: Imprime um valor na console

```
    console.log('Alô!');
```

Para realizar a entrada de dados há várias possibilidades. A seguir indicamos um exemplo com uso do módulo *prompt*.

```
var prompt = require('prompt');
  prompt.start();

  prompt.get(['usuario', 'email'], function (erro, resultado) {
    if (erro) { return onErr(erro); }
    console.log('Entrada pela console recebida:');
    console.log('  Usuário: ' + resultado.usuario);
```

```
    console.log('  Email: ' + resultado.email);
  });

  function onErr(erro) {
    console.log(erro);
    return 1;
  }
```

Certamente isso não resolve todos os casos, mas é o que vamos utilizar por enquanto.

2.4 Inserindo Código Javacript

O código Javascript poder ser inserido em diversas posições no código HTML:

- **Código Javascript no HEAD:**

```
<!DOCTYPE html>
<html>
<head>
<script>
    function myFunction() {
        document.getElementById("demo").innerHTML =
        "Parágrafo modificado.";
    }
</script>
</head>
<body>
    <h2>JavaScript no Head</h2>
    <p id="demo">Um paragráfo.</p>
    <button type="button" onclick="myFunction()">
    Clique Aqui</button>
</body>
</html>
```

- **Código Javascript no BODY:**

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript no  Body</h2>
    <p id="demo">Um parágrafo.</p>
    <button type="button" onclick="myFunction()">
    Clique Aqui.</button>
    <script>
```

```
    function myFunction() {
        document.getElementById("demo").innerHTML =
        "Paragrafo Modificado.";
    }
    </script>
</body>
</html>
```

- **Código Javascript inserido a partir de um arquivo:**

```
<!DOCTYPE html>
<html
<body>
    <h2>JavaScript Externo</h2>
    <p id="demo">Um Paragrafo.</p>
    <button type="button" onclick="myFunction()">
    Clique Aqui</button>
    <p>(myFunction está armazenado em um
    arquivo externo de nome "myScript.js")</p>
    <script src="myScript.js"></script>
</body>
</html>
```

2.5 Escrevendo no Documento HTML

A seguir alguns exemplos de como alterar um documento HTML com uso de Javascript.

Exemplo para escrever em um documento HTML:

```
<!DOCTYPE html>
<html>
<body>
    <h2>Minha Primeira Página WEB.</h2>
    <p>Meu primeiro parágrafo.</p>
    <script>
    document.write(5 + 6);
    </script>
</body>
</html>
```

Exemplo para escrever em um Elemento HTML:

```
<!DOCTYPE html>
<html>
<body>
    <h2>Minha Primeira Página WEB</h2>
    <p>Meu Primeiro Parágrafo.</p>
    <p id="demo"></p>
    <script>
    document.getElementById("demo").innerHTML = 5 + 6;
    </script>
</body>
</html>
```

Exemplo para escrever em uma Janela de Alerta:

```
<!DOCTYPE html>
<html>
<body>
    <h2>Minha Primeira Página WEB</h2>
    <p>Meu primeiro parágrafo.</p>
    <script>
    window.alert(5 + 6);
    </script>
</body>
</html>
```

Escrevendo na Console:

```
<!DOCTYPE html>
<html>
<body>
    <h2>Ative a depuração com a tecla F12</h2>
    <p>Selecione "Console" no menu de depuração.
    Então clique executar novamente.</p>
    <script>
    console.log(5 + 6);
    </script>
</body>
</html>
```

2.6 Variáveis e Comentários

2.6.1 Variáveis

O nomes em JavaScript são sensíveis ao caixa alto ou baixo: *minhaVariavel* é diferente de *minhavariavel*. A seguir diferentes formas de declarar e atribuir valor a uma variável.

```
    var minhaVariavel;
    minhaVariavel = 'Beto';
    var minhaVariavel = 'Beto';
    minhaVariavel = 'Esteves';
```

2.6.2 Comentários

Exemplo de comentários:

```
/*
Tudo o que estiver aqui é comentário
*/

// Isto também é um comentário
// que pode ser feito desta maneira
// se você quiser
```

3

Operadores e Tipos de Dados

3.1 Operadores

A seguir apresentamos alguns dos operadores mais utilizados na linguagem Javascript. Em particular, notem a diferença entre == e ===, onde o segundo operador exige que as variáveis sendo comparadas sejam do mesmo tipo para que a igualdade seja verdadeira e no primeiro caso não.

Operadores Relacionais	**Significado**
== ou ===	Igualdade (**do mesmo tipo**)
!= ou **!==**	Desigualdade (**do mesmo tipo**)
<, <=	Menor / Menor ou Igual
>, >=	Maior / Maior ou Igual

Operadores Lógicos	**Significado**
&&	E (AND)
||	OU (OR)
!	Negação (NOT)

Operadores Bit a Bit	**Significado**
&	E (AND)
|	OU (OR)
^	Ou Exclusivo (XOR)
~	Negação (NOT)

Tabela 3.1: Operadores Aritméticos

Operadores Aritméticos	**Significado**	**Exemplo**
+, -	Soma / Subtração	a+b
*, /, **	Multiplicação / Divisão / Potência	x * 2; Soma / 3; x = x ** y
%	Resto da divisão	Soma % 3
++, - -	Incremento / Decremento	a++; ++a; b- -; - -b;
=	Atribuição Simples	Media = (a+b+c) / 3;
+=, -=, *=, /=, %=, « =, »=	Atribuição Composta	A -= 1 ; equivalente a A = A - 1; R %= 2; equivalente a R = R % 2;
«, »	Deslocamento à esquerda/direita	x = x « y x = x » y

3.2 Tipos de Dados

Nesta seção relacionamos alguns tipos de dados de uso mais comum em Javascript.

Tabela 3.2: Tipos de Dados

Tipo	**Explanação**	**Exemplo**
constant	Um valor apenas de leitura, de qualquer tipo definido abaixo.	const myConstant = 20;
string	Uma sequência de texto entre plics ou aspas.	var myVariable = 'Beto' let myVariable = "Beto"
number	Os números em javascript são todos reais.	var myVariable = 10;
boolean	Um valor verdadeiro/falso, não precisa de aspas.	var myVariable = true;
array	Uma estrutura que permite armazenar diversos valores em uma única referência	var myVariable = [1,'Beto','José',10]; Cada membro do array como myVariable[0], myVariable[1]...
object	Basicamente tudo em JavaScript é um objeto e pode ser armazenado em uma variável.	myVariable = document.querySelector('h1');

3.2.1 Arrays

Arrays são um tipo de dados muito importantes em qualquer linguagem. Uma maneira de criar e declarar *arrays* em Javascript pode ser a seguinte:

```
var a = new Array();
a[0] = 'cao';
a[1] = 'gato';
a[2] = 'frango';
a.length; // 3
```

Uma notação mais conveniente é usar um *array* de literais:

```
var a = ['cao', 'gato', 'frango'];
a.length; // 3
```

Se você acessar um índice de *array* que não existe, o valor *undefined* será retornado:

```
typeof a[90]; // undefined
```

Você pode iterar sobre os elementos de um *array* usando um laço do tipo *for* e o método *a.length*, que retorna o tamanho de um *array*:

```
var a=[0,1,2,3,4,5,6,7,8,9,0];
for (var i = 0; i < a.length; i++) {
    console.log(a[i]);
}
```

Um outro modo de iterar sobre os elementos de um array foi adicionado recentemente à definição da linguagem:

```
a=[9,8,7,6,5,4,3,2,1];
a.forEach (function(temp, i, novo) {
    console.log(a[i]);
    console.log(temp);
    console.log(novo);
});
```

Se você quiser adicionar um elemento ao *array* simplesmente use o seguinte método pré-definido na linguagem:

```
a.push(item);
```

3.2.2 Métodos Pré-definidos para Arrays

- **a.toString()**: Retorna uma cadeia com a toString() de cada elemento separadas por vírgula.

- **a.toLocaleString()**: Retorna uma cadeia com a toLocaleString() de cada elemento separadas por vírgula.
- **a.concat(item1[, item2[, ...[, itemN]]])**: Retorna um novo array com os novos itens adicionados ao antigo.
- **a.join(sep)**: Converte o array em uma cadeia – com os valores delimitados com o parâmetro *sep*.
- **a.pop()**: Remove e retorna o último ítem.
- **a.push(item1, ..., itemN)**: Adiciona uma ou mais itens ao final do array.
- **a.reverse()**: Inverte o array.
- **a.shift()**: Remove e retorna o primeiro item.
- **a.slice(start[, end])**: Retorna um sub-array.
- **a.sort([cmpfn])**: Ordena o array com o uso de uma uma função de comparação (opcional) passada como parâmetro.
- **a.splice(start, delcount[, item1[, ...[, itemN]]])**: Permite que você modifique um array pagando uma seção dele e substituindo com outros itens.
- **a.unshift(item1[, item2[, ...[, itemN]]])**: Adiciona itens ao início do array.

3.3 Conversão de Tipos

3.3.1 Conversão Implícita de Tipos

Em expressões envolvendo valores numéricos e *strings* com o operador +, os valores numéricos serão convertidos implicitamente para strings. Por exemplo:

```
X = "A resposta é " + 35; // retorna "A resposta é 35"
Y = 35 + " é a resposta" // retorna "35 é a resposta"
```

Em expressões envolvendo outros operadores a conversão será feita de acordo com o tipo de dado esperado pelo operador. Exemplo:

```
X = "37" - 7 // retorna 30
Y = "37" + 7 // retorna 377
```

Quando dois valores são comparados, números, *strings* e valores lógicos são comparados por valor:

```
3 == "3" // Resultado true
1 && true // Resultado true
```

Usando a conversão implícita podemos usar de um artifício simples para forçar a conversão de número para string e de string para número.

- **Número para string:**

```
i = 25 ;
s = "" + i ; // s recebe "25"
```

- **String para número:**

```
s = "256" ;
j = s - 0 ; // j recebe 256
```

3.3.2 Conversão de Explícita de Tipos

parseInt (str) ou parseInt (str,base): Converte uma string num número inteiro. Str é a cadeia a ser convertida e base é a base em que os números estão codificados. Se a base não for especificada, assume que a string é uma seqüência de algarismos na base decimal (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Caso encontre algum caractere diferente dos esperados, encerra a conversão:

```
num = "3A";
x = parseInt(num);     // x recebe 3
y = parseInt(num,16);     // y recebe 58
```

parseFloat (str): É a função equivalente a parseInt, para converter strings em números reais. Da mesma forma, se encontrar algum caractere não esperado, encerra a conversão.

```
z = parseFloat("3.15"); // z recebe 3.15
```

4

Sentenças Condicionais e Laços

4.1 if-then-else

O uso de sentenças condicionais, assim como em outras linguagens, também é possível em Javascript. A seguir um exemplo que apresenta o resultado na console.

```
if (time < 10) {
    saudacao = "Bom dia!";
} else if (time < 20) {
    saudacao = "Boa tarde!";
} else {
    saudacao = "Boa noite!";
}
console.log(saudacao);
```

...

```
var name = 'gatinho';
if (name == 'toto') {
    name += ' auau';
} else if (name == 'gatinho') {
    name += ' miau'
} else {
    name += '!';
}
console.log(name);
```

4.2 Laços com while e for

Para iterar diversas vezes podem ser utilizados o *for* e o *while*. Inicialmente, vejamos um exemplo com *while*:

```
while (true) {
  // um laço infinito!
}

var input;
do {
  input = get_input();
} while (inputIsNotValid(input));
```

E agora exemplo com *for*:

```
for (var i = 0; i < 5; i++) {
  // vai executar 5 vezes
}

for (let value of array) {
  // faça alguma coisa com o valor
}
for (let property in object) {
  // faça alguma coisa com a propriedade do objeto

}
```

4.3 Switch

O comando *switch* pode ser usado para fazer uma escolha entre diversos itens, sem que seja necessário o uso do *if-then-else*.

```
switch (sinal) {
    case 'verde':
        Siga();
        break;
    case 'vermelho':
        Pare();
        break;
    case 'amarelo':
        Reduza();
        break;
  default:
        Pare();
}
```

5

Eventos

Eventos em HTML sao "coisas"que acontecem com os elementos HTML. Quando o Javascript é utilizado em páginas HTML, ele pode "reagir"a esses eventos. Um evento HTML pode ser alguma copisa que o navegador faz, pode ser algo que o usuário faz.

Veja a seguir alguns exemplos de eventos em HTML:

- Uma página HTML que acabou de carregar;
- Um campo de entrada HTML que mudou de valor;
- Um botão HTML que foi pressionado.

Frequentemente, quando um evento acontece, você pode querer fazer alguma coisa. O JavaScript permite que você execute código quando eventos são detectados. Veja a seguir:

```
document.querySelector('html').onclick = function() {
    alert('Ai! Não me cutuque!');
}
```

Há muitas maneiras de conectar um evento com um elemento. Aqui o elemento HTML foi selecionado, atribuindo ao manipulador *onclick* a uma função anônima, que contém o código que queremos executar. Note que

```
document.querySelector('html').onclick = function() {};
```

é equivalente a

```
var myHTML = document.querySelector('html');
myHTML.onclick = function() {};
```

O HTML permite que atributos de manipuladores de evento, com código Javascript, sejam adicionados aos elementos HTML.

Com aspas simples: **<element event='codigo JavaScript'>**

Com aspas duplas: **<element event="codigo JavaScript" >**

No exemplo seguinte, um atributo *onclick* (com código) é adicionado ao elemento *button* em HTML:

```
<button onclick="document.getElementById('demo').innerHTML =
Date()">Que horas são?</button>
```

No exemplo acima, o Javascript muda o conteúdo do elemento com **id="demo"**. No próximo exemplo, o código muda o conteúdo de seu próprio elemento (usando this.innerHTML):

```
<button onclick="this.innerHTML = Date()">Que horas são?</button>
```

O código JavaScript frequentemente tem várias linhas de código. É cada vez mais comum vermos atributos de eventos chamando funções:

```
<button onclick="displayDate()">
Que horas são?</button>
```

5.1 Elementos HTML Usuais

A seguir uma lista de alguns elementos HTML usuais:

Evento	**Descrição**
onchange	Uma mudança em um elemento HTML
onclick	O usuário clicou em um elemento HTML
onmouseover	O usuário movimentou o mouse sobre o elemento HTML
onmouseout	O usuário moveu o mouse para fora do elemento HTML
onkeydown	O usuário pressionou alguma tecla no teclado
onload	O navegador terminou de carregar a página

O manipuladores de evento podem ser utilizados para manipular e verificar a entrada de dados do usuário, ações do usuário e ações do navegador:

- Coisas que devem ser feitas toda vez que uma página é carregada;
- Coisas que devem ser feitas quando uma página é fechada;
- Ações que devem ser realizadas quando um usuário clica um botão;
- Conteúdo que deve ser verificado quando o usuário faz uma entrada de dados.
- E mais...

6

Objetos

Há duas formas básicas para se criar um objeto vazio:

```
var obj = new Object();
```

```
var obj = {};
```

Essas formas são semanticamente equivalentes e a segunda é chamada sintaxe literal do objeto e é mais conveniente. Esta sintaxe também é o núcleo do formato JSON e deve ser preferida todas as vezes. A sintaxe literal do objeto pode ser usada para iniciar um objeto em sua totalidade.

```
var obj = {
  name: 'Cenoura',
   for: 'Mario',
  details: {
    color: 'laranja',
    size: 12
  }
}
```

Atributos dos objetos podem ser encadeados juntos:

```
obj.details.color; // orange
obj['details']['size']; // 12
```

Os exemplos a seguir criam um protótipo de um objeto, **Pessoa**, e uma instância deste protótipo, **Voce**.

```
function Pessoa(nome, idade) {
  this.nome = nome;
  this.idade = idade;
```

```
}
// Define um objeto
var Voce = new Pessoa('Voce', 24);
// Estamos criando uma nova pessoa de nome  "Voce"
// com a idade igual a 24
```

Uma vez criado, as propriedades de um objeto podem ser criadas de dois modos:

```
obj.nome = 'Simonal';
var nome = obj.nome;
E ...
obj['nome'] = 'Simonal';
var nome = obj['nome'];
```

Essas formas também são semanticamente equivalentes. O segundo método tem a vantagem que o nome da propriedade é fornecido como uma string, o que significa que ela pode ser calculada em tempo de execução.

Contudo, o uso deste método previne que algumas máquinas Javascript utilizem certas otimizações de minimização. Esta forma também pode ser usada para atribuir propriedades com nomes que são palavras reservadas no Javascript:

```
obj.for = 'Simonal'; // Erro de sintaxe, porque 'for' é uma
                     // palavra reservada
obj['for'] = 'Simonal'; // Funciona sem problemas
```

7

Funções

Uma função JavaScript pode ter 0 ou mais parâmetros nomeados. O corpo da função pode conter tantas declarações quanto você quiser, e variáveis que são locais para aquela função podem ser declaradas no corpo da função.

A declaração *return* pode ser usada para retornar um valor a qualquer momento, terminando a função.

```
function soma(num1,num2) {
  var resultado = num1 + num2;
  return resultado;
}

soma(4,7);
soma(20,20);
soma(0.5,3);
```

Se nenhuma declaração de retorno for usada (ou um *return* vazio sem nenhum valor), o JavaScript retornará o valor *undefined*.

Os parâmetros enumerados revelam-se mais como orientações do que qualquer outra coisa. Você pode chamar uma função sem passar os parâmetros que espera, caso em que eles serão definidos como *undefined*.

```
soma(); // NaN
// Não se pode realizar a soma em um undefined
```

Você também pode passar mais argumentos do que a função está esperando:

```
soma(2, 3, 4); // 5
// somou os dois primeiros argumentos; 4 foi ignorado
```

7.1 Argumentos de Funções

Isso pode parecer um pouco simples, mas as funções têm acesso a uma variável adicional dentro de seu corpo chamada *argument*, que é um objeto do tipo *array* contendo todos os valores passados para a função. Vamos re-escrever a função soma para receber quantos valores quisermos:

```
function soma() {
    var resultado = 0;
for (var i = 0, j = arguments.length; i < j; i++) {
      resultado += arguments[i];
      }
return resultado;
}
soma(2, 3, 4, 5); // 14
```

Isso realmente não é mais útil do que apenas escrever 2 + 3 + 4 + 5. Vamos criar então uma função que calcule a média:

```
function media() {
var soma = 0;
      for (var i = 0, j = arguments.length; i < j; i++) {
            soma += arguments[i];
        }
  return soma / arguments.length;
  }
  media(2, 3, 4, 5); // 3.5
```

7.2 Operador Spread

O operador *spread* é usado em declarações de função com o formato: ... [variável] e incluirá dentro dessa variável toda a lista de argumentos não capturados com os quais a função foi chamada. Vamos também substituir o laço *for* por um laço *for ... of* para retornar os valores dentro de nossa variável.

```
function media(...argumentos) {
    var soma = 0;
    for (let valor of argumentos) {
        soma += valor;
    }
    return soma / argumentos.length;
}
```

```
    media(2, 3, 4, 5); // 3.5
```

7.3 Funções Anônimas

O JavaScript permite que você crie funções anônimas.

```
var avg = function() {
  var sum = 0;
  for (var i = 0, j = arguments.length; i < j; i++) {
    sum += arguments[i];
  }
  return sum / arguments.length;
};
```

7.4 Recursividade em Funções

O JavaScript permite que você chame funções recursivamente. Isso é particularmente útil para lidar com estruturas de árvore, como as encontradas no DOM do navegador.

```
function countChars(elm) {
  if (elm.nodeType == 3) { // TEXT_NODE
    return elm.nodeValue.length;
  }
  var count = 0;
  for (var i = 0, child; child = elm.childNodes[i]; i++) {
    count += countChars(child);
  }
  return count;
}
```

Observe que as funções JavaScript são objetos - como tudo o mais em JavaScript - e você pode adicionar ou alterar suas propriedades, como vimos anteriormente.

7.5 Funções anexadas a Objetos

Vamos criar duas funções para exibir o nome de uma pessoa. Basicamente há duas maneiras pelas quais o nome pode ser exibido: como "primeiro último"ou como "último, primeiro". Usando as funções e objetos que discutimos anteriormente, poderíamos exibir os dados como este:

```
function criaPessoa(primeiro, ultimo) {
  return {
    primeiro: primeiro,
    ultimmo: ultimo,
    Nomecompleto: function() {
      return this.primeiro + ' ' + this.ultimo;
    },
    NomecompletoReverso: function() {
      return this.ultimo + ', ' + this.primeiro;
    }
  };
}
s = criaPessoa('Wilson' ,'Simonal');
s.Nomecompleto(); // "Wilson Simonal"
s.NomecompletoReverso(); // "Simonal, Wilson"
```

Podemos aproveitar a palavra-chave *this* para melhorar nossa função **criaPessoa**:

```
function Pessoa(primeiro, ultimo) {
  this.primeiro = primeiro;
  this.ultimo = ultimo;
  this.Nomecomplero = function() {
    return this.primeiro + ' ' + this.ultimo;
  };
  this.NomecompletoReverso = function() {
    return this.ultimo + ', ' + this.primeiro;
  };
}
var s = new Pessoa('Wilson','Simonal');
```

Introduzimos outra palavra-chave: *new*. Ela cria um novo objeto vazio e, em seguida, chama a função especificada, com este conjunto para esse novo objeto. Por exemplo:

```
function Pessoa(primeiro, ultimo, idade, olhos) {
    this.primeiroNome = primeiro;
    this.ultimoNome = ultimo;
    this.idade = idade;
    this.corOlhos = olhos;
    this.nome = function() {return this.primeiroNome + " " + this.
   ultimoNome;};
}
var s = new Pessoa('Wilson','Simonal', 18, 'blue');
console.log(s.primeiroNome);
console.log(s.ultimoNome);
console.log(s.nome());
```

Esta função pode ainda ser melhorada criando-se as funções do método apenas uma vez, e atribuindo referências a elas dentro do construtor.

```
function pessoaNomeCompleto() {
  return this.primeiro + ' ' + this.ultimo;
}
function pessoaNomeCompletoReverso() {
  return this.ultimo + ', ' + this.primeiro;
}
function Pessoa(primeiro, ultimo) {
  this.primeiro = primeiro;
  this.ultimo = ultimo;
  this.Nomecompleto = pessoaNomeCompleto;
  this.NomecompletoReverso = pessoaNomeCompletoReverso;
}
```

7.5.1 Protótipos de Funções

Podemos também criar um protótipo de função, que depois pode ser utiilzado para criar diversos tipos de objetos.

```
function Pessoa(primeiro, ultimo) {
  this.primeiro = ultimo;
  this.ultimo = ultimo;
}
Pessoa.prototype.Nomecompleto = function() {
  return this.primeiro + ' ' + this.ultimo;
};
Pessoa.prototype.NomecompletoReverso = function() {
  return this.ultimo + ', ' + this.primeiro;
};
var s = new Pessoa('Wilson','Simonal');
console.log(s.primeiro);
console.log(s.ultimo);
console.log(s.Nomecompleto());
console.log(s.NomecompletoReverso());
```

8

Node.js

Node.js é uma plataforma para desenvolvimento de aplicações "server-side" baseadas em rede utilizando JavaScript e o V8 JavaScript Engine, ou seja, com Node.js podemos criar uma variedade de aplicações Web utilizando apenas código em JavaScript. Tem as seguintes características:

- Consistente: as representações de linguagem e dados do servidor/cliente são as mesmas
- Escalável: Arquitetura com uma única thread minimiza o uso de memória e evita custos de mudança de contexto entre threads
- Rápido (em determinadas coisas)

O Node.js é especialmente útil se a E/S for provavelmente o seu gargalo (ou seja, o servidor não está fazendo muita coisa ...). Exemplos: entradas em fila, streaming de dados, websockets... De um modo geral, o Node.js é ideal para tarefas leves e em tempo real e uma má escolha para tarefas computacionalmente intensivas. O seu site pode ser encontrado no endereço: http://www.nodejs.org

Veja um exemplo simples [1] seguir:

```
var http = require('http');
 http.createServer(function(req,res) {
   res.writeHead(200, { 'Content-Type':
   'text/plain; charset=utf-8' });
   res.end('Olá mundo!\n');
 }).listen(3000);
 console.log('Servidor iniciado em
 localhost:3000. Ctrl+C para encerrar...');
```

Que pode ser executado com os comandos a seguir:

[1] https://tableless.com.br/o-que-nodejs-primeiros-passos-com-node-js/

```
$ node arquivo.js
Servidor iniciado em localhost:3000. Ctrl+C para encerrar...
```

Em outro terminal executar

```
$ curl http://localhost:3000/
> Olá mundo!
```

8.1 npm

O Node Package Manager (npm) fornece duas funcionalidades principais:

- Repositórios online para pacotes e módulos node.js que podem ser localizados em http://search.nodejs.org
- Utilitários de linha de comando para instalar pacotes node.js, fazendo o gerenciamento de versões e dependências.

O npm é instalado junto com o node.js. Para verificar qual a sua versão, abra a console (ou execute cmd no windows) e digite o seguinte comando:

```
$ npm -v
 4.3.0
```

Se você está rodando uma versão antiga do npm, então é muito fácil atualizar para a última versão, usando o seguinte comando:

```
$ sudo npm install npm -g
/usr/bin/npm  -> /usr/lib/node_modules/npm/bin/npm-cli.js
npm@2.7.1 /usr/lib/node_modules/npm
```

Para iniciar um novo projeto use o seguinte comando:

```
$ mkdir teste
$ cd teste
$ npm init -y
```

Este comando irá criar um arquivo package.json com a descrição do projeto que terá o mesmo nome que o diretório corrente.

Qualquer um dos comandos a seguir listam todos os pacotes instalados.

```
$ npm list
$ npm ls
```

Para remover todos os pacotes que o seu projeto não dependa, segundo o informado pelo arquivo package.json, use o comando:

```
$ npm prune
```

Para saber quais pacotes instalados estão desatualizados, de acordo com o que está registrado pelo npm, e a versão definida no arquivo package.json, use o comando:

```
$ npm outdated
```

Se agora quisermos instalar o módulo chamado "Johnny-Five" localmente podemos usar o comando:

```
$ npm install --save  johnny-five
```

Isso irá criar um diretório node_modules no diretório atual onde o pacote johnny-five será instalado juntamente com todos os pacotes com quem tenha dependências.

A opção - -save faz com que o arquivo package.json seja alterado para indicar que o pacote johnny-five já foi instalado localmente. O comando:

```
$ npm search mkdir
```

Verifica se tem algum pacote com nome ou que realize funções similares de manipulação e diretório. O comando:

```
$ npm uninstall johnny-five
```

Pode ser utilizado para desinstalar o pacote johnny-five junto com as suas dependências.
Na realidade há duas maneiras de instalar as coisas:

- globalmente – isso coloca os módulos em {prefix}/lib/node_modules, e coloca os arquivos executáveis em {prefix}/bin, onde {prefix} usualmente é algo como /usr/local. Isso também instala as páginas de manual em {prefix}/share/man, se houver.
- localmente – Isso instala seu pacote no diretório de trabalho atual em ./node_modules, executáveis vão para ./node_modules/.bin/, e as páginas de manual não são instaladas.

Pacotes e dependências instalados globalmente são armazenados em um diretório do sistema. Em geral, a regra padrão é a seguinte:

Se você está instalando que você deseja usar apenas no seu programas, usando require ('pacote'), então instale-o localmente, na raiz do seu projeto.

Se você está instalando alguma coisa que você quer usar no seu shell, ou na linha de comando ou algo assim, instale-o globalmente, de modo que os binários possam ser encontrados no caminho definido pela variável de ambiente PATH. Vamos tentar instalar o modulo johnny-five utilizando uma instalação global:

Isso irá produzir um resultado similar mas o módulo será instalado globalmente. A primeira linha da saída amostra a versão do módulo e o lugar onde está sendo instalado.

8.2 Gerenciando a Cache

Para atualizar um pacote que já está instalado, o seguinte comando pode ser utilizado:

```
$ npm update johnny-five
```

Quando o npm instala um pacote ele guarda uma cópia deste pacote, de modo que na próxima vez que você precisar deste pacote, não seja necessário ir à internet novamente.

O diretório onde esta cópia é guardada é /.npm no Posix, ou %AppData%/npm-cache no Windows.

Eventualmente este diretório pode ficar cheio com pacotes velhos, então é necessário que ele seja limpo de vez em quando com o comando:

```
$ npm cache clean
```

Para utilizar um módulo no seu arquivo js, inclua a linha como a seguir:

```
var express  = require ('prompt');
```

As regras de onde "require" encontra os arquivos podem ser um pouco complexas, mas uma regra simples é que se o arquivo não começa com "./" ou "/", então ele é considerado um módulo global (e o caminho de instalação do "Node" local é verificado), ou é uma dependência local na pasta "node_modules" do projeto local.

Se o arquivo começa com "./" é considerado um arquivo relativo ao diretório do arquivo que chamou "require". Se o arquivo começa com "/", ele é considerado um caminho absoluto.

OBSERVAÇÃO: você pode omitir a extensão ".js" e a função "require" irá automaticamente anexá-la se necessário.

Veja um exemplo bem simples a seguir:

```
var prompt = require('prompt');
```

```
prompt.start();
prompt.get(['usuario', 'email'], function (err, result) {
  if (err) { return onErr(err); }
  console.log('Entrada na linha de comando recebida:');
  console.log('  Usuário: ' + result.usuario);
  console.log('  Email: ' + result.email);
});
function onErr(err) {
  console.log(err);
  return 1;
}
```

9

JSON

O JSON (JavaScript Object Notation) é um formato de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar. Está baseado em um subconjunto da linguagem de programação JavaScript.

O JSON é apresentado em formato texto e completamente independente de linguagem, pois usa convenções que são familiares a diversas linguagens, incluindo C, C++, C#, Java, JavaScript, Perl, Python, entre outras. O JSON é constituído de duas estruturas:

- Uma coleção de pares nome/valor. Em várias linguagens, isto é caracterizado como um objeto, registro, estrutura, dicionário, etc.
- Uma lista ordenada de valores. Na maioria das linguagens, isto é caracterizado como uma arranjo, vetor, lista ou sequência.

São estruturas de dados universais. Virtualmente todas as linguagens de programação modernas as suportam, de uma forma ou de outra. É aceitável que um formato de troca de dados que seja independente de linguagem de programação se baseie nestas estruturas. Em JSON, os dados são apresentados desta forma:

- Um objeto é um conjunto desordenado de pares nome/valor.
- Um objeto começa com { (chave de abertura) e termina com } (chave de fechamento).
- Cada nome é seguido por : (dois pontos) e os pares nome/valor são seguidos por , (vírgula).

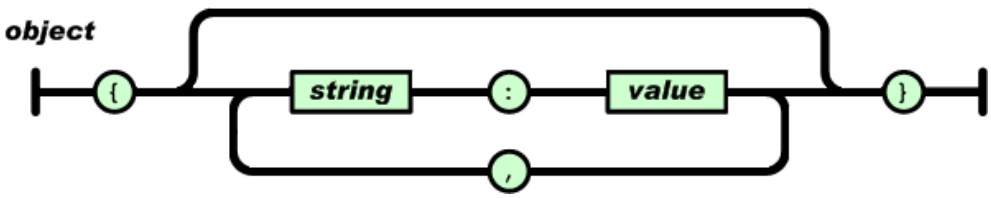

Figura 9.1: object

- Um *array* é uma coleção de valores ordenados. O *array* começa com [(colchete de abertura) e termina com] (colchete de fechamento). Os valores são separados por , (vírgula).

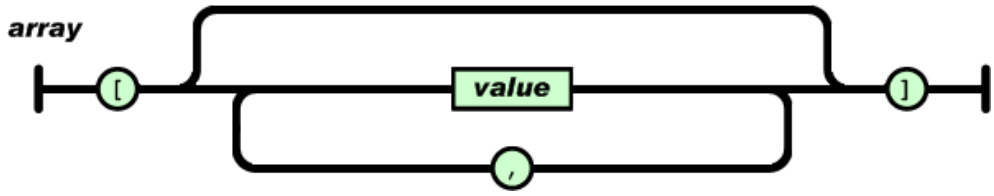

Figura 9.2: array

- Um valor pode ser uma cadeia de caracteres (*string*), ou um número, ou *true ou false*, ou *null*, ou um objeto ou um *array*. Estas estruturas podem estar aninhadas.

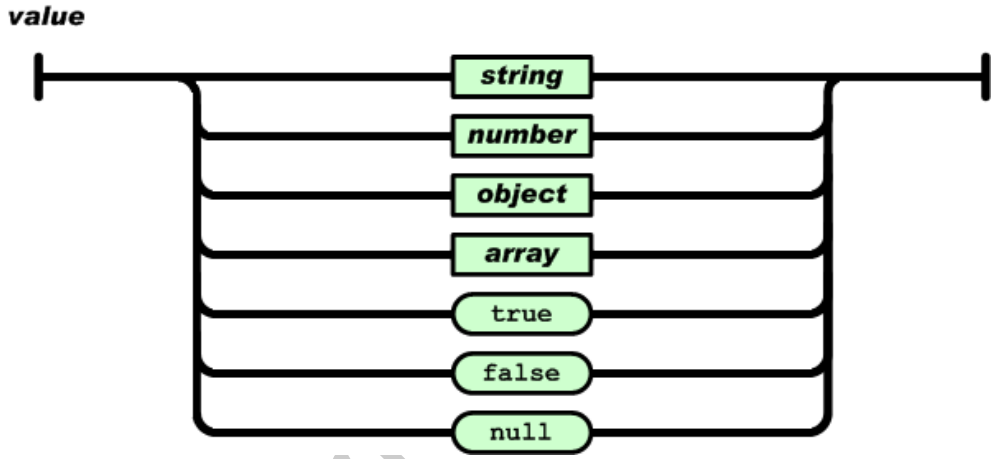

Figura 9.3: value

- Uma *string* é uma coleção de nenhum ou mais caracteres unicode, entre aspas duplas, usando barras invertidas como caractere de escape. Um caractere é representado como um simples caractere de *string*. Uma cadeia de caracteres é parecida com uma cadeia de caracteres em C ou Java. (Figura 9.4)

- Um número é similar a um número em C ou Java, exceto que os números octais ou hexadecimais não são usados. (Figura 9.5)

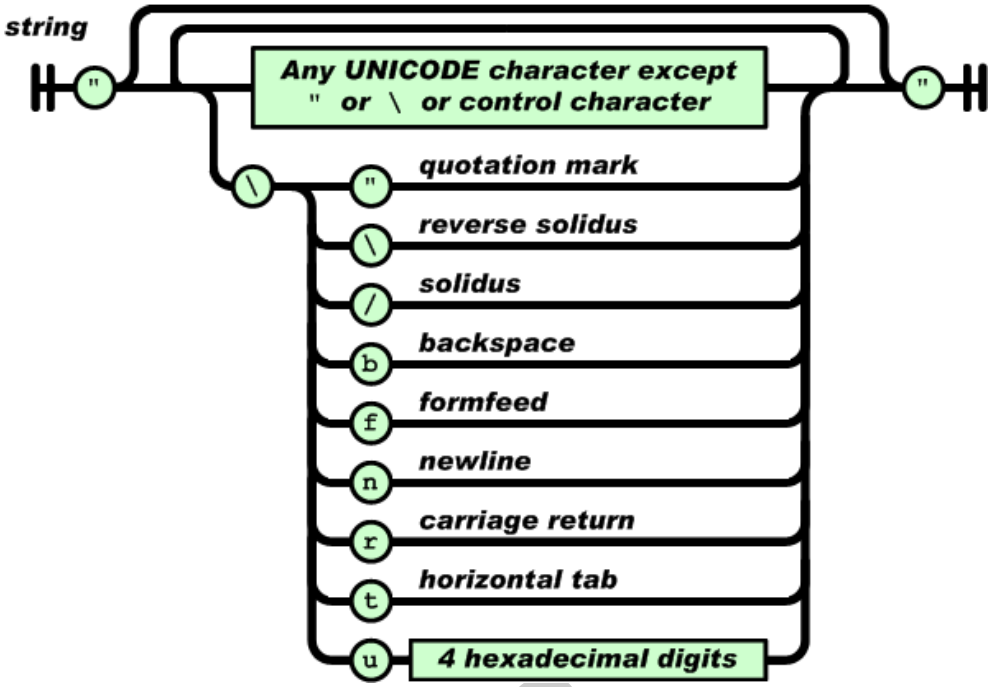

Figura 9.4: string

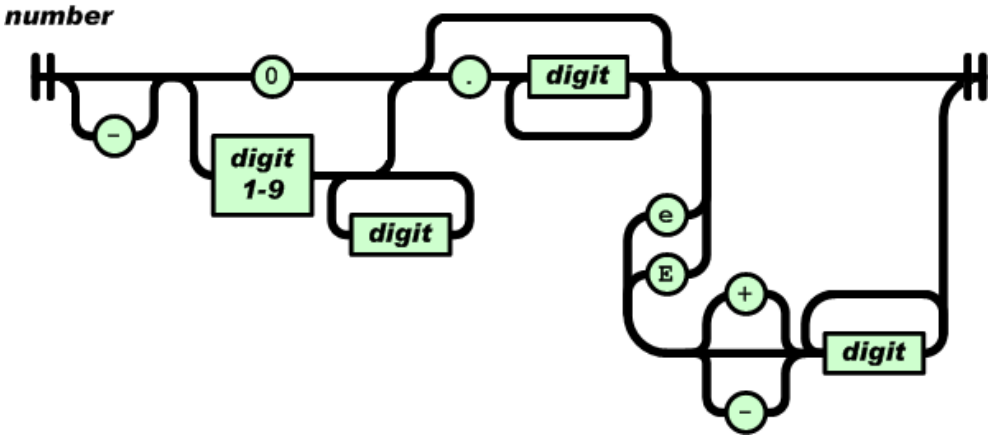

Figura 9.5: number

```
object
    {}
    { members }
members
    pair
    pair , members
pair
    string : value
array
    []
    [ elements ]
elements
    value
    value , elements
value
    string
    number
    object
    array
    true
    false
    null
string
    ""
    " chars "
chars
    char
    char chars
char
    any-Unicode-character-
        except-''-or-\-or-
        control-character
    \''
    \\
    \/
    \b
    \f
    \n
    \r
    \t
    \u four-hex-digits
number
    int
    int frac
    int exp
    int frac exp
```

```
int
    digit
    digit1-9 digits
    - digit
    - digit1-9 digits
frac
    . digits
exp
    e digits
digits
    digit
    digit digits
e
    e
    e+
    e-
    E
    E+
    E-
```

Referências

1. Mozilla Developer Network - https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript
2. Node.js - NPM - https://www.tutorialspoint.com/nodejs/nodejs_npm.htm
3. Introducing JSON - http://www.json.org