

**UNIR**

**UNIVERSIDADE FEDERAL  
DE RONDÔNIA**

**DEPARTAMENTO DE FÍSICA**

**Simulação de Fenômenos Físicos em JavaScript**

**JS**

**Prof. Antonio F. Cardozo**

## ① Documento HTML

## ① JavaScript

## ② HTML5 Canvas

### Objetivos

Apresentar os conceitos e comandos básicos do Núcleo da Linguagem **Java Script**

### Objetivos Operacionais

Levar ao aluno a adquirir habilidades para desenvolver e executar simulação de fenômenos físicos da mecânica clássica tendo como recurso a linguagem de programação Java Script

## ① Documento HTML

### 0.2 Tabelas

### 0.3 Formulários

## ① JavaScript

### 1.2 Código JavaScript

- 1.2.1 Operadores aritméticos
- 1.2.2 Operadores de Comparação
- 1.2.3 Operadores Bit a bit
- 1.2.4 Operadores de Atribuição
- 1.2.5 Operadores Lógicos
- 1.2.6 Estruturas de Controle
- 1.2.7 switch ... case
- 1.2.8 while
- 1.2.9 do ... while
- 1.2.10 for
- 1.2.11 for ... in
- 1.2.12 Arrays Associativos

### 1.3 Funções

- 1.3.1 Expressão function
- 1.3.2 Funções Internas

### 1.4 Objetos em Javascript

- 1.4.1 Como criar propriedades
  - 1.4.2 Métodos em um objeto
- Recursos para interação com os

### 1.5 usuários

- 1.5.1 Eventos MouseDown, MouseUp
- 1.5.2 Funções para interação do usuário

### 1.5.3 Event

### 1.5.4 Exemplo MouseDown e MouseUp

### 1.5.5 Loop infinito em JavaScript

### 1.5.6 Função timedCount()

### 1.5.7 Saída de dados com alert ()

### 1.5.8 Entrada de dados com prompt()

## ② HTML5 Canvas

### 2.1 Ex Canva1

### 2.2 Ex Canva2

### 2.3 O sistema de coordenadas do Canvas

### 2.4 Desenhar figuras geométricas em Canvas

### 2.5 Propriedades fillStyle, strokeStyle e lineWidth

### 2.6 context. beginPath (); (caminhos

### 2.7 método stroke

## ③ Codificação de problemas Físicos

### 3.1 Exemplo Indicador de Graus

### 3.2 Aplicação com a funçãoTimeCout

### 3.3 Rotação ctx.rotate

### 3.4 Exemplo Rotat translate

### 3.5 Rotação Matriz

## ④ Programas

### 4.1 Sistema Massa Mola Amortecido


## 🕒 Documento HTML

### Estrutura básica de um documento HTML

Um documento **HTML** recebe algumas tags que formam a sua estrutura básica. No HTML5, o documento padrão recebe a seguinte estrutura:

```
<!DOCTYPE html>
<html>
<head>
  <title>Título da página</title> <meta charset="utf-8"/>
</head>
<body>
</body>
</html>
```

## tags:

**<!DOCTYPE html>** – A tag **!DOCTYPE** informa ao navegador a versão do HTML que está sendo utilizada no documento. Por exemplo: no HTML5, basta incluir **!DOCTYPE html**, e assim o navegador já saberá que se trata de um documento na versão HTML5; 

**<html></html>** – Esta tag é o elemento básico da estrutura do HTML. Assim sendo, ela conterá todos os elementos do seu documento. Todo documento HTML deve iniciar e finalizar com essa tag;

**<head></head>** – Essa tag delimita o cabeçalho do documento. Não possui nenhum valor visível, porém é capaz de transmitir ao navegador diversas informações muito úteis e essenciais a uma boa apresentação do seu documento HTML;

**<title></title>** – Essa tag define o título da sua página, o nome que aparecerá na sua aba, janela ou guia.

**<meta/>** – Essa tag permite inserir metadados ao seu documento, no caso acima, a informação **charset="UTF-8"**, que garante a compatibilidade do código com os caracteres de padrão latino americano;

**<body></body>** – Finalmente, a tag que representa o corpo do documento. Em síntese, é nessa tag que todos os elementos visíveis do seu site devem ser inseridos.

## Tags HTML estruturais

As tags abaixo são utilizadas nos documentos em [HTML5](#), e têm função estrutural no seu código. Portanto, essas tags têm grande importância na questão semântica da página

`<header></header>` – Essas tags definem um cabeçalho. Portanto, todo conteúdo que estiver dentro dela faz parte de um cabeçalho, podendo ser utilizado dentro de outras sessões. Não confundir com as tags `<head>`;

`<main></main>` – Essas tags representam o conteúdo principal do seu corpo, ou seja, o conteúdo relacionado diretamente com o tópico central da página ou com a funcionalidade central da aplicação;

`<footer></footer>` – Essas tags definem um rodapé para a página, geralmente utilizadas no final da página;

`<section></section>` – Essas tags definem uma sessão para sua página;

`<article></article>` – Essas tags definem um artigo da sua página. Nesse sentido, são utilizadas para separar o conteúdo da sua página. Muito utilizado principalmente por blogs ou páginas de conteúdo;

`<aside></aside>` – Essas tags representam uma seção de uma página cujo conteúdo é tangencialmente relacionado ao conteúdo do seu entorno, que poderia ser considerado separado do conteúdo;

`<nav></nav>` – Essa tag define um conteúdo de navegação. Portanto, é muito utilizado em conjunto com listas e na criação de menus;

`<div></div>` – Define uma divisão da página. Desta forma, funciona como um container para conteúdo de fluxo. Uma vez que não possui um valor semântico, é muito utilizado para organizar melhor o conteúdo. Anteriormente ao HTML5, era utilizado no lugar das categorias acima.

## Tags HTML de título

As tags de título possuem valor semântico, variando entre seis níveis hierárquicos. É importante entender como funcionam, e fazer uma utilização adequada. Para definir títulos, utilizamos as tags:

`<h1></h1>` - Título de maior valor hierárquico

`<h2></h2>` `<h3></h3>` `<h4></h4>` `<h5></h5>` `<h6></h6>` - Título de menor valor hierárquico

## Tags HTML de texto

As tags de texto definem textos, estilos de fonte, parágrafos, spans, quebras de linhas, etc. Vamos conhecê-las:

`<p></p>` – Principal tag de texto, compõe um parágrafo;

`<span></span>` – Apesar de ter uma funcionalidade e características parecidas com os parágrafos, costumam ser utilizadas apenas para pequenas informações, como legendas de um formulário, legendas de uma imagem, etc. Também pode ser utilizada para formar um container;

`<pre></pre>` – Tag utilizada para representar texto pré-formatado. Muito utilizada para inserir códigos;

`<b></b>` – Transforma o conteúdo em negrito;

`<i></i>` – Transforma o conteúdo em itálico;

`<br/>` – Essa tag não necessita de fechamento, ela executa a função de quebra de linha.

`<hr/>` – Essa tag não necessita de fechamento, ela forma uma linha horizontal.



## Tag de link HTML

A tag de [link HTML](#) é responsável que faz a ligação entre um documento e outro, sendo ele da mesma página ou de uma página de outro domínio.

Para realizar um link, podemos chamar as tags `<a></a>` com o atributo href. Por exemplo, caso você queira criar um link no seu texto que redirecione à página inicial do google:

`<p>Para acessar o Google, <a href="https://www.google.com">clique aqui.</a></p>`

## Tags HTML de multimídia

As tags de multimídia servem para incluir imagens, vídeos, áudios, iframes e outros tipos de conteúdo multimídia. `<img>`

Essa tag não necessita de fechamento, serve para incluir uma imagem ao seu texto. A partir dessa tag, utilizamos o atributo `src=""` onde deve ser digitado o local em que a imagem se encontra. Também é muito utilizado em conjunto com o atributo `alt` para definir o texto alternativo da imagem. Por exemplo:

```

```

<b>Tags de Vídeos</b>	serve para indicar a inserção de um vídeo
<b>&lt;iframe&gt;</b>	servem para incluir recursos de uma outra página nesta página
<b>Tags HTML de listas</b>	podemos utilizar uma lista ordenada, a partir das tags <code>&lt;ol&gt;&lt;/ol&gt;</code> , ou uma lista não ordenada, a partir das tags <code>&lt;ul&gt;&lt;/ul&gt;</code> . Posteriormente, incluímos dentro da lista os elementos da mesma, dentro das tags <code>&lt;li&gt;&lt;/li&gt;</code> .
<b>Tags de formulário tags <code>&lt;form&gt;</code> e <code>&lt;/form&gt;</code></b>	são muito utilizadas para obter informações do usuário, realizar cadastros, receber opiniões, entre outros. São importantíssimas para qualquer ramo do mercado.
<b>tag <code>&lt;script&gt;</code> e <code>&lt;/script&gt;</code></b>	tem como objetivo incluir códigos de scripts ao seu HTML

## Accessing HTML Documents on the Web

O arquivo index.htm contém a página inicial para este indivíduo. Por padrão, o arquivo index.htm é automaticamente aberto, se existir, sempre que este URL for acessado. Ou seja, o endereço

<http://www.myUniversity.edu/~myName/>

# 1 JavaScript

## 1.2 Código JavaScript

Para nossos propósitos, um elemento script sempre contém código JavaScript. Esses elementos são organizados da seguinte forma em um documento HTML:

Editores: (Notepad++) (Brackets-2.2.1)

```
<html>
  <head>
    <title> ... </title>
    <!-- Optional script elements
    <script>
    </script>
  </head>
  <body>
  </body>
</html>
```

```
<html>
  <head>
    <title>Hello, world!</title>
    <script language="javascript"
    type="text/javascript">
      // These statements display text in
      a document.
      document.write("Hello, world!");
      document.write("<br />It's a
      beautiful day!");
    </script>
  </head>
  <body>
    <!-- No content in the body of this
    document. -->
  </body>
</html>
```

## 1.2.1 Operadores aritméticos

### Aritméticos

<b>Operador</b>	<b>Operação</b>	<b>Exemplo</b>
<b>+</b>	Adição	$x+y$
<b>-</b>	Subtração	$x-y$
<b>*</b>	Multiplicação	$x*y$
<b>/</b>	Divisão	$x/y$
<b>%</b>	Módulo (resto da divisão inteira)	$x\%y$
<b>-</b>	Inversão de sinal	$-x$
<b>++</b>	Incremento	$x++$ ou $++x$
<b>--</b>	Decremento	$x--$ ou $--x$

## 1.2.2 Operadores de Comparação

### Comparação

<b>Operador</b>	<b>Função</b>	<b>Exemplo</b>
<b>==</b>	Igual a	(x == y)
<b>!=</b>	Diferente de	(x != y)
<b>===</b>	Idêntico a (igual e do mesmo tipo)	(x === y)
<b>!==</b>	Não Idêntico a	(x !== y)
<b>&gt;</b>	Maior que	(x > y)
<b>&gt;=</b>	Maior ou igual a	(x >= y)
<b>&lt;</b>	Menor que	(x < y)
<b>&lt;=</b>	Menor ou igual a	(x <= y)

### 1.2.3 Operadores Bit a bit

#### 4.2.3 Bit a bit

<b>Operador</b>	<b>Operação</b>	<b>Exemplo</b>
<b>&amp;</b>	E (AND)	(x & y)
<b> </b>	OU (OR)	(x   y)
<b>^</b>	Ou Exclusivo (XOR)	(x ^ y)
<b>~</b>	Negação (NOT)	~x
<b>&gt;&gt;</b>	Deslocamento à direita (com propagação de sinal)	(x >> 2)
<b>&lt;&lt;</b>	Deslocamento à esquerda (preenchimento com zero)	(x << 1)
<b>&gt;&gt;&gt;</b>	Deslocamento à direita (preenchimento com zero)	(x >>> 3)

## 1.2.4 Operadores de Atribuição

### Atribuição

<b>Operador</b>	<b>Exemplo</b>	<b>Equivalente</b>
<b>=</b>	$x = 2$	Não possui
<b>+=</b>	$x += y$	$x = x + y$
<b>-=</b>	$x -= y$	$x = x - y$
<b>*=</b>	$x *= y$	$x = x * y$
<b>/=</b>	$x /= y$	$x = x / y$
<b>%=</b>	$x %= y$	$x = x \% y$
<b>&amp;=</b>	$x \&= y$	$x = x \& y$
<b> =</b>	$x  = y$	$x = x   y$
<b>^=</b>	$x ^= y$	$x = x ^ y$
<b>&gt;&gt;=</b>	$x >>= y$	$x = x >>= y$
<b>&lt;&lt;=</b>	$x <<= y$	$x = x <<= y$
<b>&gt;&gt;&gt;=</b>	$x >>>= y$	$x = x >>>= y$



## 1.2.5 Operadores Lógicos

### Lógicos

Operador	Função	Exemplo
&&	E Lógico	(x && y)
	OU Lógico	(x    y)
!	Negação Lógica	!x

```
if(x > 915 && x < 1086 && y > 430 && sinal==1)
{
    alert(x);
}
```

Operador	wwew	Descrição
AND lógico (&&)	expr1 && expr2	(E lógico) - Retorna expr1 caso possa ser convertido para falso; senão, retorna expr2. Assim, quando utilizado com valores booleanos, && retorna verdadeiro caso ambos operandos sejam verdadeiros; caso contrário, retorna falso.
OU lógico (  )	expr	Negação lógica) Retorna falso caso o único operando possa ser convertido para verdadeiro; senão, retorna verdadeiro.
NOT lógico (!)	!expr	(Negação lógica) Retorna falso caso o único operando possa ser convertido para verdadeiro; senão, retorna verdadeiro.

## 1.2.6 Estruturas de Controle

### if ... else

A estrutura if é usada quando se deseja verificar se determinada expressão é verdadeira ou não, e executar comandos específicos para cada caso.

```
if(x > 915 && x < 1086 && y > 430 && sinal==1)
{
    alert(x);
}
```

```
if (a > 0)
{
    result = 'positive';
}
else
{
    result = 'NOT positive';
}
```

```
if(xt < 0.0)
{
    if(ht > 0.0)
        Ang0 = 180 - Math.abs(Math.atan(ht / xt) * 57.324800000000003);
        form.campo22.value=Ang0.toFixed(3);
    if(ht < 0.0)
        Ang0 = 180 + Math.abs(Math.atan(ht / xt) * 57.324800000000003);
        form.campo22.value=Ang0.toFixed(3);
        AntRelógio();
}
```

Assim, se a expressão for avaliada como verdadeira, o primeiro bloco de comandos é executado. Caso seja avaliada como falsa, o bloco de comandos que segue o *else* será executado.

Também é possível aglomerar mais testes, utilizando-se o comando *else if*.

#### Exemplo 2:

```
var a = 10;
if (a < 6) {
    window.alert("a menor que 6");
} else if (a > 6) {
    window.alert("a maior que 6");
} else {
    window.alert("se a não é maior nem menor que 6, a é 6!");
}
```

## 1.2.7 switch ... Case

As estruturas do tipo switch são usadas quando queremos selecionar uma opção dentre várias disponíveis.

```
93  switch(expression) {  
94      case x:  
95          // code block  
96          break;  
97      case y:  
98          // code block  
99          break;  
100     default:  
101         // code block  
102 }
```

Ao contrário de outras linguagens, os valores de comparação podem ser strings além de valores numéricos. O comando `break` faz com que o `switch` pare de verificar as outras possibilidades abaixo e pode ser omitido caso se deseje uma estrutura que tornará mais de uma opção como verdadeira. Por fim, `default` é opcional e corresponde a uma sequência de comandos que será executada quando nenhum dos outros `case` o forem.

### 1.2.8 while

Os laços do tipo `while` são usados quando se deseja que uma sequência de ações seja executada apenas no caso da expressão de condição ser válida. Assim, primeiro a expressão é testada, para depois o conteúdo do laço ser executado ou não.

Exemplo:

```
92   var cont = [5,2];
93   while ((cont[0]+cont[1]) < 15) {
94     cont[0]+=1;
95     cont[1]+=2;
96     document.write('cont0 = '+cont[0]+'cont1 = '+cont[1]);
97   }
98   // Com o uso de while, no primeiro teste, cont[0]+cont[1] vale 7;
```

## 1.2.9 do ... while

Diferentemente do while, o do ... while primeiro executa o conteúdo do laço uma vez e, depois disso, realiza o teste da expressão para decidir se continuará executando o laço ou irá seguir o resto do programa.

Exemplo:

Execute um bloco de código uma vez e continue se a condição ( $i < 5$ ) for verdadeira:

```
93   let text = "";
94   let i = 0;
95   do {
96     text += i + "<br>";
97     i++;
98   }
99   while (i < 5)
```

## 1.2.10 for

Na maioria das vezes, quando usamos um laço do tipo while também construímos uma estrutura com um contador que é incrementado a cada passo para controle do laço e manipulação interna de objetos, arrays como nos exemplos anteriores. Os laços for oferecem a vantagem de já possuírem em sua estrutura essa variável de contador e incrementá-la de maneira implícita.

Exemplo:

```
82     for(i=1;i<=9;i+=1)
83     {
84         var n=
85             ctx.fillText(50-10*i,0,20*i-6);
86             ctx.stroke();ctx.beginPath();
87     }
```

```
72     for(i=1;i<N;i++)
73     {
74         vx[i]=0.1;
75         vy[i]=0.1;
76         Fx[i]=0.1;
77         Fy[i]=0.1;
78         x[i]=0.1;
79         y[i]=0.1;
80         LP=0;
81         t1=0;
82         t2=0;
83         F=0;
84     }
```



## 1.2.12 Arrays Associativos

### *Arrays*

Um *array* é uma coleção de elementos. *Arrays* em JavaScript não são um *tipo* próprio. *Arrays* são **objetos**. Podemos inicializar um *array* vazio de 2 maneiras diferentes:

```
const a = []  
const a = Array()
```

A primeira usa uma **sintaxe literal**. A segunda utiliza uma função construtora. Você pode pré-preencher o *array* usando esta sintaxe:

```
const a = [1, 2, 3]  
const a = Array.of(1, 2, 3)
```

Um *array* pode conter qualquer valor, mesmo valores de tipos diferentes:

```
const a = [1, 'Flavio', ['a', 'b']]
```

Como podemos incluir um *array* dentro de um *array*, podemos criar *arrays* multidimensionais, que possuem aplicações muito úteis (por exemplo, uma matriz):

```
const matriz = [  
  [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9]  
]  
matriz[0][0] //1  
matriz[2][0] //7
```

## Como exibir uma Matriz

A maneira mais fácil e recomendada de exibir uma matriz, é usando dois laços for, aninhados. No primeiro laço, uma variável percorre as linhas. No segundo laço, o interno, outra variável percorre o número correspondente de cada coluna.

```
9      function ff()  
10     {  
11  
12     const mat=[  
13         [1, 2, 3],  
14         [4, 5, 6],  
15         [7, 8, 9]  
16     ]  
17     alert(mat[0][0]); //1  
18     alert(mat[2][0]); //7  
19  
20     }
```

## Como Declarar uma Matriz

```
208   var q = [26];
209
210   q[0]=0.792;q[1]=0.928;q[2]=1.064;q[3]=1.2;q[4]=1.336;q[5]=1.472;q[6]=1.608;q[7]=1.744;q[8]=1.88;q[9]=2.016;q[10]=2.152;
211   q[11]=2.288;q[12]=2.424;q[13]=2.56;q[14]=2.696;q[15]=2.832;q[16]=2.968;q[17]=3.104;q[18]=3.24;q[19]=3.376;q[20]=3.512;
212   q[21]=3.648;q[22]=3.784;q[23]=3.92;q[24]=4.056;q[25]=4.192;
213
214   //alert( Q[0] ) retorna 0.792
```

## Como exibir uma Matriz

```
9  function ff()
10  {
11  form.textarea.value += "m  " + "v  "+'\r\n';
12
13      var Matrix_A = [[0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0],
14                      [0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0],
15                      [0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0],];
16
17      for(i=0;i<11;i++)
18      {
19          for (p=0;p<11;p++)
20          {
21              Matrix_A[i][p]=i+p;
22          }
23      }
24      for(i=0;i<11;i++)
25      {
26          for (p=0;p<11;p++)
27          {
28              form.textarea.value += "[i][p]= " + Matrix_A[i][p]+' \r\n';
29          }
30      }
```

## 1.3 Funções

Funções são blocos de construção fundamentais em JavaScript. Uma função é um procedimento de JavaScript - um conjunto de instruções que executa uma tarefa ou calcula um valor. Para usar uma função, você deve defini-la em algum lugar no escopo do qual você quiser chamá-la.

### Declarando uma função

A **definição da função** (também chamada de **declaração de função**) consiste no uso da palavra chave [function \(en-US\)](#), seguida por:

- Nome da Função.
- Lista de argumentos para a função, entre parênteses e separados por vírgulas.
- Declarações JavaScript que definem a função, entre chaves { }.

```
12 function f1(n)
13     {
14         return Math.cos(n);
15     }
16
17 function f2(n)
18     {
19         return n*n;
20     }
```

### 1.3.1 Expressão function

A primeira maneira de se declarar uma função é através do uso da palavra chave function de maneira similar a como elas são declaradas na linguagem C, com as diferenças de que em JavaScript não definimos o tipo de retorno e nem mesmo o tipo dos argumentos.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Apostila JavaScript Progressivo</title>
  </head>
  <body>
    <script type="text/javascript">
      document.write("O quadrado de 2 é: ");
      document.write( quadrado(2) );

      function quadrado(numero)
      {
        var resposta = numero * numero;
        return resposta;
      }
    </script>
  </body>
</html>
```

## 1.3.2 Funções Internas

O Que é o Math no JavaScript?

**Math** é um objeto que acompanha o JavaScript puro. Ele é utilizado para realizar operações matemáticas, tais como: operações aritméticas, funções trigonométricas, funções de arredondamento e comparações.

É importante saber que o objeto Math não é um construtor e todas as suas propriedades e métodos são estáticos. Você pode referenciar a constante PI como **MATH.PI** e você pode chamar a função de seno como **MATH.SIN(X)** onde **X** é o argumento do método. Constantes são definidas com a precisão total de números reais em JavaScript.

1	<a href="#">Math.E</a>	Constante de Euler e base dos logaritmos naturais, aproximadamente 2.718.
2	<a href="#">Math.LN2</a>	Logaritmo natural de 2, aproximadamente 0.693.
3	<a href="#">Math.LN10</a>	Logaritmo natural de 10, aproximadamente 2.303.
4	<a href="#">Math.LOG2E</a>	Logaritmo de E na base 2, aproximadamente 1.443.
5	<a href="#">Math.LOG10E</a>	Logaritmo de E na base 10, aproximadamente 0.434.
6	<a href="#">Math.Pi</a>	Relação entre a circunferência de um círculo e o seu diâmetro, proximadamente 3.14159.
7	<a href="#">Math.SQRT1_2</a>	Raiz quadrada de 1/2; Equivale a 1 dividido pela raiz quadrada de 2, aproximad. 0.707.
8	<a href="#">Math.SQRT2</a>	Raiz quadrada de 2, aproximadamente 1.414.
9	<a href="#">Math.abs(x)</a>	Retorna o módulo, ou valor absoluto, de um número ()
10	<a href="#">Math.acos(x)</a>	Retorna o arco-coseno de um número (arccos).
11	<a href="#">Math.acosh(x)</a>	Retorna o arco-coseno hiperbólico de um número.
12	<a href="#">Math.asin(x)</a>	Retorna o arco-seno de um número (
13	<a href="#">Math.asinh(x)</a>	Retorna o arco-seno hiperbólico de um número
14	<a href="#">Math.atan(x)</a>	Retorna o arco-tangente de um número (
15	Math.atanh(x	Retorna o arco-tangente hiperbólico de um número (
16	<a href="#">Math.atan2(x, y)</a>	Retorna o arco-tangente do quociente de seus argumentos.
17	<a href="#">Math.cbrt(x)</a>	Retorna a raiz cúbica de um número (
18	<a href="#">Math.ceil(x)</a>	Retorna o menor inteiro que é maior ou igual a um número.
19	<a href="#">Math.cos(x)</a>	Retorna o coseno de um número (
20	<a href="#">Math.cosh(x)</a>	Retorna o coseno hiperbólico de um número .
21	<a href="#">Math.exp(x)</a>	é a constante de Euler (2.718...), a base do logaritmo natural. $e^x$ onde $x$ é o argumento, e
22	<a href="#">Math.expm1(x)</a>	Retornam $e^x+1$
23	<a href="#">Math.floor(x)</a>	Retorna o maior inteiro que é menor ou igual a um número.



24	<a href="#">Math.fround(x) (en-US)</a>	Retorna a mais próxima representação de ponto flutuante de <a href="#">precisão-única</a> de um número
25	<a href="#">Math.hypot([x[,y[,...]]])</a>	Retorna $\sqrt{x^2 + y^2}$
26	<a href="#">Math.imul(x) (en-S)</a>	Retorna o resultado de uma multiplicação de inteiro de 32-bit.
27	<a href="#">Math.log(x)</a>	Retorna $\log_e x$
28	<a href="#">Math.log1p(x)</a>	Retorna $\log_e (x + 1)$
29	<a href="#">Math.log10(x)</a>	Retorna o logaritmo de x na base 10 (
30	<a href="#">Math.log2(x)</a>	Retorna o logaritmo de x na base 2
31	<a href="#">Math.max([x[,y[,...]]])</a>	Retorna o maior dentre os parâmetros recebidos.
32	<a href="#">Math.min([x[,y[,...]]])</a>	Retorna o menor dentre os parâmetros recebidos.
33	<a href="#">Math.pow(x,y)</a>	Retorna a base x elevada à potência y
34	<a href="#">Math.random()</a>	Retorna um número pseudo-aleatório entre 0 e 1.
35	<a href="#">Math.round(x)</a>	Retorna o valor arredondado de x, para o valor inteiro mais próximo.
36	<a href="#">Math.sign(x)</a>	Retorna o sinal de x, indicando se é positivo, negativo ou zero.
37	<a href="#">Math.sin(x)</a>	Retorna o seno de um número (
38	<a href="#">Math.sinh(x)</a>	Retorna o seno hiperbólico de um número (
39	<a href="#">Math.sqrt(x)</a>	Retorna a raiz quadrada positiva de um número (
40	<a href="#">Math.tan(x)</a>	Retorna a tangente de um número (
41	<a href="#">Math.tanh(x)</a>	Retorna a tangente hiperbólica de um número (
42	<a href="#">Math.toSource()</a>	Retorna a string "Math".
43	<a href="#">Math.trunc(x)</a>	Retorna a parte inteira de x, removendo quaisquer dígitos racionários

## 1.4 Objetos em Javascript

CommandBu

### Introdução

Objeto em Javascript é uma estrutura na qual você pode juntar várias variáveis e depois acessar essas variáveis de forma independente. As variáveis internas pode ser de qualquer tipo, incluindo funções e outros objetos.

É comum chamar as variáveis internas de um objeto de *propriedades*. As variáveis que são funções são chamadas de métodos. Objeto é a forma como fazemos orientação a objetos em Javascript

### Como declarar um objeto

Um objeto em Javascript precisa estar envolto em chaves “{}”. O código abaixo criar um objeto vazio:

```
1 let obj = {}
```

## 1.4.1 Como criar propriedades

Existem várias forma de criarmos propriedades em um objetos. Primeiramente podemos criar junto com a definição do objeto. Neste caso cada propriedade tem que ser inserida no formato *nome: valor*. As propriedade também dever ser separadas por vírgulas, como no exemplo abaixo:

```
1 let circle = {  
2   radius: 50,  
3   x: 100,  
4   y: 200,  
5   color: "red"  
6 }
```

Vemos que a separação entre o nome e o valor deve usar o dois pontos ":" e não o igual "=".

## Como acessar as propriedades

Para acessar as propriedades de um objeto existem duas formas, a “*brackets notation*” e a “*dot notation*”. O “*dot notation*” (notação de ponto) usa um ponto entre o nome do objeto e o nome da propriedade. Se quisermos então imprimir no console o raio do círculo criado no objeto acima, podemos fazer assim:

```
1 console.log(circle.radius) dot notation
```

Isso imprime o valor 50 no terminal. Vemos então que os objetos organizam muito o código, pois podemos colocar variáveis relacionadas em um lugar só, acessando-as através do seu objeto. Isso é conceito de **orientação a objetos**, onde criamos um objeto com propriedades próprias, variáveis ou funções, e esse objeto encapsula sua lógica própria.

Já com a “bracket notation” colocamos o nome do objeto e o nome da propriedade entre colchetes. Seguindo o exemplo anterior, temos:

```
1 console.log(circle["radius"]) brackets notation
```

## 1.4.2 Métodos em um objeto

Como em Javascript funções são elementos de primeira ordem, podemos colocar funções, em variáveis, portanto podemos ter funções dentro de objetos, como no exemplo abaixo:

```
1 let circle = {  
2   radius: 50,  
3   x: 100,  
4   y: 200,  
5   color: "red",  
6   draw: function() {  
7     console.log("I'm a circle!")  
8   }  
9 }
```

A propriedade “draw” que é uma função. Podemos chamar a função usando o “dot notation”, como abaixo:

```
1 circle.draw()
```

Ou seja, chamamos a “draw” como uma função normal, mas agora ela está dentro de um objeto.

A palavra chave *this*

A palavra chave *this* em Javascript é muito especial. Ela significa algo um pouco diferente dependendo da situação e farei um vídeo no futuro detalhando todos esses casos. Porém no contexto de um objeto, usamos o *this* dentro de uma função de um objeto para acessar as outras propriedades do objeto do qual a função faz parte.

No exemplo do círculo acima podemos fazer com que a função *draw* desenhe um círculo usando o raio e as coordenadas x e y do objeto. Fica então assim:

```
1 let circle = {
2   radius: 50,
3   x: 100,
4   y: 200,
5   color: "red",
6   draw: function() {
7     ellipse(this.x, this.y, this.radius, this.radius);
8   }
9 }
```

Vemos então que dentro do *draw* usamos o *this* para acessar as outras variáveis do objeto. Quando você coloca o *this*, o Javascript entende que você está acessando as outras variáveis que fazem parte do objeto do qual a função sendo executada faz parte. Sem o *this* não teria como a função saber o que é o *x*, *y* e *radius*.

O *this* então é como se fosse um ponteiro ou referência para o objeto.

## Criando propriedades dinamicamente com o “dot notation”

Podemos também criar novas propriedades em um objeto já existem apenas especificando o nome e valor com a notação de ponto, e caso a propriedade não existir ela é criada com o nome e valor passados. Por exemplo, se quisermos adicionar um método *update* no objeto acima, podemos criar da seguinte forma:

```
1 circle.update = function(x,y) {  
2     this.x = x;  
3     this.y=y;  
4 }
```

O código acima cria uma nova propriedade *update* que atualiza as propriedades *x* e *y* do objeto. Portanto podemos criar propriedades dinamicamente com Javascript. Isso difere de outras linguagens orientadas a objetos, como C++, onde temos as classes que são estáticas, ou seja, depois que um objeto é instânciado não podemos criar novas propriedades. Mas em Javascript isso é possível.

## Exemplo sobre referência ao **this**

A referência ao **this** sempre se refere a (e contém o valor de) um objeto — um objeto singular — e normalmente é usado dentro de uma função ou método, embora possa ser usado fora de uma função no escopo global.

```
1 var person = {
2   firstName : "Penelope",
3   lastName  : "Barrymore",
4   // Já que a palavra-chave "this" é usada dentro do método showFullName() abaixo e
5   // é definido no objeto "person", "this" terá o valor do objeto "person" porque e
6   // invocará showFullName().
7   showFullName : function() {
8     console.log ( this.firstName + ' ' + this.lastName );
9   }
10 }
11
12 person.showFullName(); // Penelope Barrymore
```



## Exemplo em jQuery

```
1 $( 'button' ).on( 'click', function( event ) {  
2     // $( this ) terá o valor do objeto "button" porque ele invoca o clique  
3     console.log ( $( this ).prop( 'name' ) );  
4 } );
```

## 1.5 Recursos para interação com os usuários

### 1.5.1 Eventos MouseDown, MouseUp

A sequência de eventos relacionados ao mouse é: Ordem dos eventos para os botões esquerdo e central do mouse

1. MouseDown
2. MouseUp
3. Click
4. DbClick
5. MouseUp

Ordem dos eventos para o botão direito do mouse:

1. onmousedown
2. onmouseup
3. oncontextmenu

## 1.5.2 Funções para interação do usuário

```
24 <script>
25   var canvas = document. getElementById ( "tcanvas" );
26   var context = canvas. getContext ( "2d" );
27
28   canvas. addEventListener ( "mousedown" , function (me) {mDown (me)}, false );
29   canvas. addEventListener ( "mousemove" , function (me) {mMove (me)}, false );
30   canvas. addEventListener ( "mouseup" , function (me) {mUp (me)}, false );
31   canvas. addEventListener ( "mouseout" , function (me) {mOut (me)}, false );
32
33 </script>
```

## 1.5.3 Event

Event	Ocorre quando
onclick	O usuário clica em um elemento
oncontextmenu	O usuário clica com o botão direito em um elemento
ondblclick	O usuário clica duas vezes em um elemento
onmousedown	Um botão do mouse é pressionado sobre um elemento
onmouseenter	O ponteiro é movido para um elemento
onmouseleave	O ponteiro é movido para fora de um elemento
onmousemove	O ponteiro está se movendo sobre um elemento
onmouseout	O ponteiro do mouse sai de um elemento
onmouseover	O ponteiro do mouse é movido sobre um elemento
onmouseup	O botão do mouse é liberado sobre um elemento

## 1.5.4 Exemplo MouseDown e MouseUp

```
1  <! DOCTYPE html>
2  <html lang = "kr" >
3    <head><meta charset = "utf-8" > </head>
4    <body>
5      <canvas id = "tcanvas" width = "1000" height = "200" style = ' border : 1px solid #000 ' ; > </canvas>
6      <script>
7        var canvas = document. getElementById ( "tcanvas" );
8        var context = canvas. getContext ( "2d" );
9
10     canvas. addEventListener ( "mousedown" , function (me) {mDown (me)}, false );
11     canvas. addEventListener ( "mousemove" , function (me) {mMove (me)}, false );
12     canvas. addEventListener ( "mouseup" , function (me) {mUp (me)}, false );
13     canvas. addEventListener ( "mouseout" , function (me) {mOut (me)}, false );
14     var stX =0;
15     var stY =0;
16     var drag = false ;
17     function mDown(me)
18     {
19       stX = me. offsetX ;
20       stY = me. offsetY ;
21       form.campo21.value=me.button;
22       drag = true ;
23     }
```

## Mause uP

## Código

```
1  <!DOCTYPE html>
2  <html lang = "kr" >
3  <head><meta charset = "utf-8" > </head>
4  <body>
5  <canvas id = "tcanvas" width = "1000" height = "200" style = ' border : 1px solid #000 ' ; > </canvas>
6  <script>
7  var canvas = document. getElementById ( "tcanvas" );var canvas = document. getElementById ( "tcanvas" );
8  canvas. addEventListener ( "mousedown" , function (me) {mDown (me)}, false );
9  canvas. addEventListener ( "mousemove" , function (me) {mMove (me)}, false );
10 canvas. addEventListener ( "mouseup" , function (me) {mUp (me)}, false );
11 canvas. addEventListener ( "mouseout" , function (me) {mOut (me)}, false );
12 var stX =0; var stY =0;
13 var drag = false ;
14 function mDown(me)
15 {
16     stX = me. offsetX ; stY = me. offsetY ;
17     form.campo21.value=me.button;drag = true ;
18 }
19 function mMove(me)
20 {
21     if (!drag)
22     {
23         return ;
24     }
25     var nowX = me. offsetX ;var nowY = me. offsetY ;canvasDraw (nowX,nowY);
26     stX = nowX; stY = nowY;
27     form.campo22.value="move";
28 }
```

```
29     function canvasDraw(currentX,currentY)
30     {
31     context. beginPath ();
32     context. moveTo (stX,stY);
33     context..lineTo (currentX,currentY);
34     context. stroke ();
35     }
36
37     function mUp(me)
38     {
39     form.campo22.value="liverado"
40     drag = false ;
41     }
42     function mOut(me)
43     {
44     drag = false ;
45     }
46     </script>
47     <form name="form">
48     <INPUT TYPE="button" value="Calcular" onClick='desenhar()'>
49     "mDown" <input name="campo21" formnovalidate value="1"><br>
50     "mUp" <input name="campo22" formnovalidate value="10">
51     </form>
52     </body>
53 </html>
```

## 1.5.5 Loop infinito em JavaScript

- Infinite Loop Using **while** Loop in JavaScript
- Infinite Loop Using **for** Loop in JavaScript
- Infinite Loop Using **timedCount()** in JavaScript

the <b>while</b> loop	<pre>let i = 0 while (i &lt; 3) {   console.log(i); i += 1; }</pre>
Infinite Loop Using <b>for</b>	<pre>for (let i = 0; i &lt; 3; i++) {   console.log(i) }</pre>
function <b>timedCount()</b>	<pre>function timedCount() {   t=setTimeout("timedCount()",freq); }</pre>



## 1.5.6 Função timedCount()

### Exemplo de TimeCount

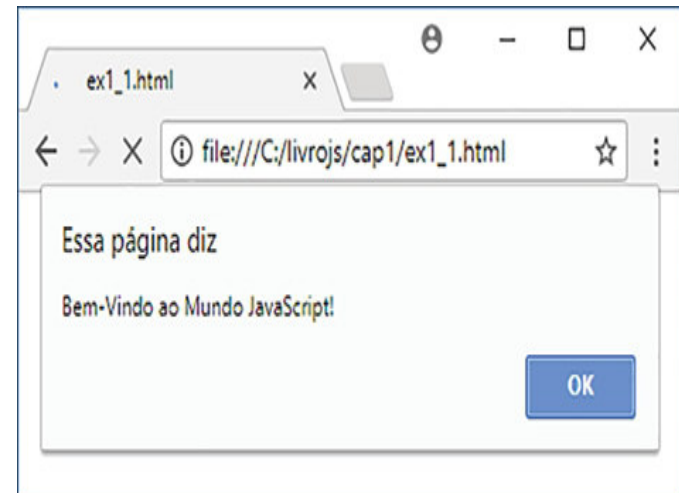
```
32  function timedCount()
33  {
34      c=c+0.1;
35      t=setTimeout("timedCount()",freq);
36      form.campo3.value=c;
37  }
38
39  function Para()
40  {
41      clearTimeout(t);
42  }
```

## 1.5.7 Saída de dados com alert (ex1\_1.html)

Começamos com algo bem simples: apresentar uma mensagem ao usuário. Na tela em branco do Visual Studio Code, informe os seguintes comandos:

Exemplo 1.1 – Saída de dados com alert (ex1\_1.html)

```
<script>  
alert("Bem-Vindo ao Mundo JavaScript!");  
</script>
```

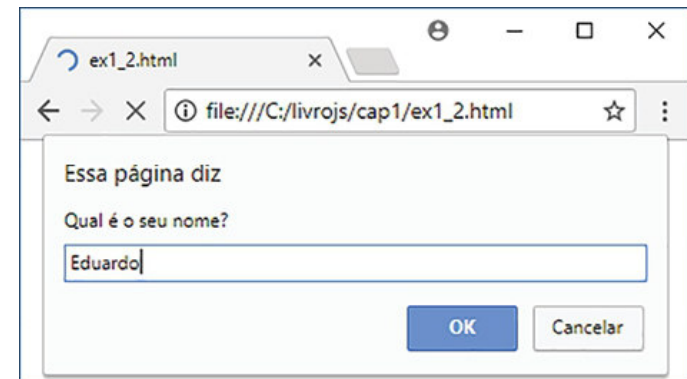


## 1.5.8 Entrada de dados com prompt()

Vamos avançar um pouco? Já apresentamos uma mensagem na tela em nosso primeiro exemplo. Vamos agora receber uma informação e apresentar uma mensagem utilizando a informação recebida. Para isso, vamos utilizar o conceito de variável visto na seção anterior e aprender um novo comando JavaScript. Para receber dados do usuário, uma das formas possíveis em JavaScript é utilizar o comando (método) `prompt()`, que exibe uma caixa com um texto e um espaço para digitação. Crie um segundo programa, com os códigos do Exemplo 1.2.

Entrada de dados e uso de variáveis (ex1\_2.html)

```
<meta charset="UTF-8">
<script>
var nome = prompt("Qual é o seu nome?");
alert("Olá " + nome);
</script>
```



## ② HTML5 Canvas

O Canvas é uma área retangular em uma página web onde podemos criar desenhos programaticamente, usando JavaScript (a linguagem de programação normal das páginas HTML). Com esta tecnologia, podemos criar trabalhos artísticos, animações e jogos

- `<canvas id="nome_canvas" width="largura" height="altura"> </canvas>`
- `<canvas id="canvas" width="700" height="500"></canvas>`

## 2.1 Ex Canva1

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Canvas API</title>
  </head>
  <body>
    <canvas id ="myCanvas1" width="800" height="400" style="border:2px solid #FF0000;"></canvas>
    <script>

    </script>
    <form name="form">
      <INPUT TYPE="button" value="Calcular" onClick='desenhar() '>
      Teste <input name="campo21" formnovalidate value="1">
      angulo <input name="campo22" formnovalidate value="10">
    </form>
  </body>
</head>
</html>
```

## 2.2 Ex Canva 2

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Canvas API</title>
  </head>
  <body>
    <canvas id="myCanvas1" width="800" height="400" style="border:2px solid #FF0000;"></canvas>
    <script>
      function f()
      {
        var canvas = document.getElementById('myCanvas1');// Canvas e contexto
        var context = canvas.getContext('2d');
        // Preenchimento vermelho
        context.fillStyle = 'red';
        context.fillRect(50, 50, 100, 100);
        // Contorno azul, com espessura de 3px
        context.lineWidth = 3;
        context.strokeStyle = 'blue';
        context.strokeRect(50, 50, 100, 100)
      }
    </script>
    <form name="form">
      <INPUT TYPE="button" value="Calcular" onClick='f()'>
      Teste <input name="campo21" formnovalidate value="1">
      angulo <input name="campo22" formnovalidate value="10">
    </form>
  </body>
</html>
```

## 2.3 O sistema de coordenadas do Canvas

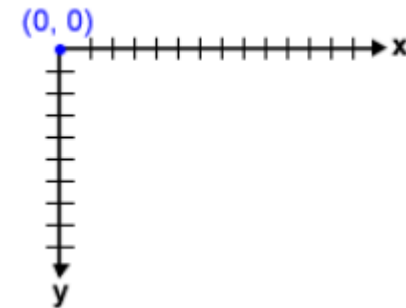
Para posicionarmos os desenhos no Canvas, pensamos nele como um enorme conjunto de pontos. Cada ponto possui uma posição horizontal (x) e uma vertical (y). O ponto (0, 0) (lê-se: zero em x e zero em y) corresponde ao canto superior esquerdo do Canvas:

```
<!DOCTYPE html>
<html>

<head>
  <title>Processando o Canvas após a tag</title>
</head>

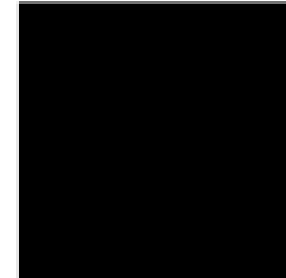
<body>
  <canvas id="meu_canvas" width="200" height="200"></canvas>
  <script>
    // Aqui obteremos o contexto gráfico e trabalharemos com o
    // Canvas
  </script>
</body>

</html>
```

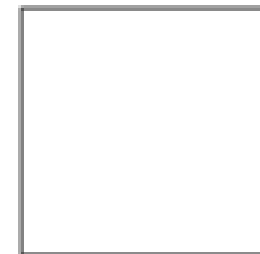


## 2.4 Desenhar figuras geométricas em Canvas

```
1 <canvas id="meu_canvas" width="200" height="200"></canvas>
2 <script>
3 // Canvas e contexto
4 var canvas = document.getElementById('meu_canvas');
5 var context = canvas.getContext('2d');
6
7 // Desenhar um retângulo
8 context.fillRect(50, 50, 100, 100);
9 </script>
```



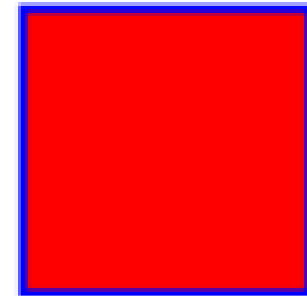
Se trocarmos `fillRect` por `strokeRect`, veremos apenas o contorno:





## 2.5 Propriedades fillStyle, strokeStyle e lineWidth

```
1 <script>
2 // Canvas e contexto
3 var canvas = document.getElementById('meu_canvas');
4 var context = canvas.getContext('2d');
5
6 // Preenchimento vermelho
7 context.fillStyle = 'red';
8 context.fillRect(50, 50, 100, 100);
9
10 // Contorno azul, com espessura de 3px
11 context.lineWidth = 3;
12 context.strokeStyle = 'blue';
13 context.strokeRect(50, 50, 100, 100);
14 </script>
```



## 2.6 context. beginPath (); (caminhos)

Desenhos mais complexos podem ser desenhados como paths (caminhos). Um path é um conjunto de comandos de desenho que ficam registrados na memória, aguardando os métodos fill (preencher) ou stroke (contornar) serem chamados.

Porém, antes de tudo, devemos chamar o método beginPath (iniciar caminho) para apagar os traçados feitos previamente. Se não fizermos isso, eles ficarão na memória e serão desenhados novamente junto com o próximo path:

## 2.7 método stroke

Esses comandos não desenham as linhas imediatamente, apenas armazenam as informações no path. Devemos chamar o método stroke para desenhá-las de fato através do moveTo e.lineTo.

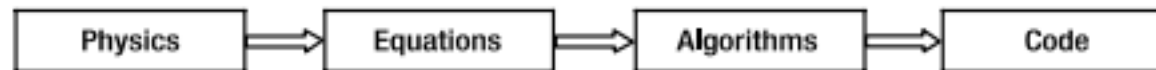
```
1 <canvas id="meu_canvas" width="300" height="300"></canvas>
2 <script>
3     // Canvas e contexto
4     var canvas = document.getElementById('meu_canvas');
5     var context = canvas.getContext('2d');
6     // Iniciar o caminho (apaga desenhos anteriores)
7     context.beginPath();
8     // Desenhar uma estrela
9     context.moveTo(75, 250); // Ponto inicial
10    context.lineTo(150, 50);
11    context.lineTo(225, 250);
12    context.lineTo(50, 120);
13    context.lineTo(250, 120);
14    context.lineTo(75, 250);
15    // Configurar a linha
16    context.lineWidth = 2;
17    context.strokeStyle = 'red';
18    // Traçar as linhas do caminho
19    context.stroke();
20 </script>
```



### 3. Codificação de problemas físicos

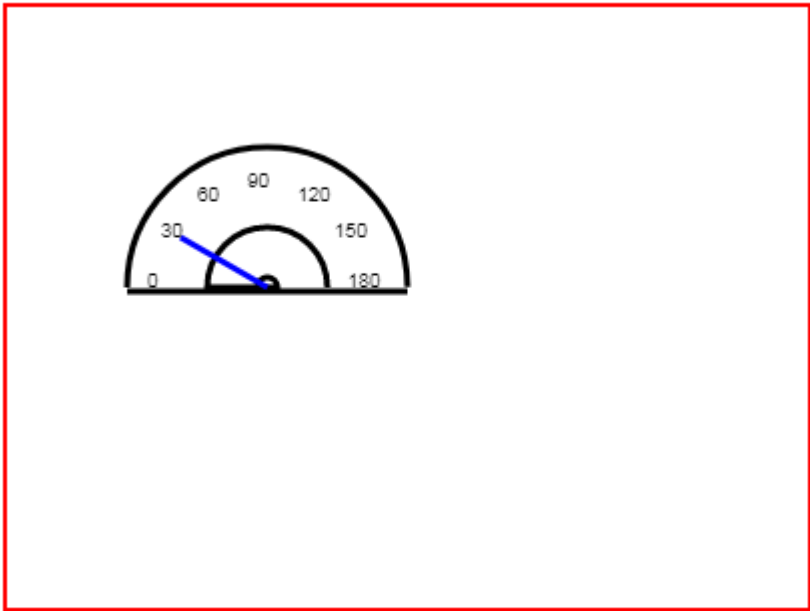
Depois de conhecer um pouco de física básica (e um pouco de matemática relevante), codificá-la não será muito diferente ou mais difícil do que você está acostumado como programador, desde que faça isso da maneira certa. O caminho certo é descrever o que está envolvido na simulação da física real e como isso é feito através de etapas que envolve m equações matemáticas, algoritmos e código.

As quatro etapas para programar física Para responder à terceira pergunta que fizemos no início desta seção, o processo de programação da física pode ser dividido em quatro etapas, conforme mostrado esquematicamente na Figura 1-1.



### 3.1 Exemplo Indicador de Graus

Código



desenhar

Angulo

## 3.2 Exemplo Time Pressão

Código

Minicurso JavaScript



Executar

Parar

Limpar

Freq

100

36

# Rotação

## 3.3 Rotação ctx.rotate

Example

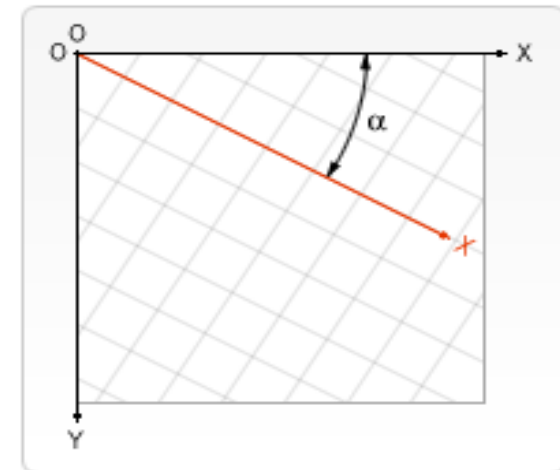
Rotate the rectangle 20 degrees:

JavaScript:

```
const canvas =  
document.getElementById("myCanvas");  
const ctx = canvas.getContext("2d");  
ctx.rotate(20 * Math.PI / 180);  
ctx.fillRect(50, 20, 100, 50);
```

O segundo método de transformação é o `rotate()`. Usamos ele para rotacionar o *canvas* em torno da origem atual. `rotate(angle)` em radianos.

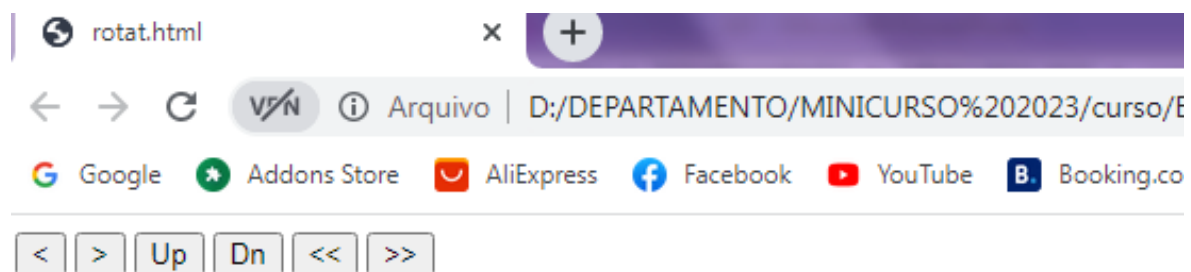
O ponto central de rotação sempre é o ponto de origem. Para alterar o ponto central, precisamos mover o canvas usando o método `translate()`.



### 3.4 Exemplo Rotat Translate

#### Código

```
cx.translate(c.x, c.y); cx.rotate(Math.PI / (180.0 / this.rect.a));
```





### 3.5 Rotação por Matriz

Rotação de um ponto em relação à origem

O exemplo acima nos permite girar um ponto no eixo  $x$  em relação à origem. Mas e se ele não estiver no eixo  $x$ ? Isso requer trigonometria um pouco mais avançada. Se chamarmos a distância entre o ponto  $(x, y)$  e a origem de  $r$ , e o ângulo entre a linha para  $(x, y)$  e o eixo  $x$  de  $\alpha$ , então

$$x = r \times \cos(\alpha) \quad \text{e} \quad y = r \times \sin(\alpha)$$

Se girarmos mais  $\beta$  até o ponto  $(x', y')$ , então:

$$x' = r \times \cos(\alpha + \beta) \quad \text{e} \quad y' = r \times \sin(\alpha + \beta)$$

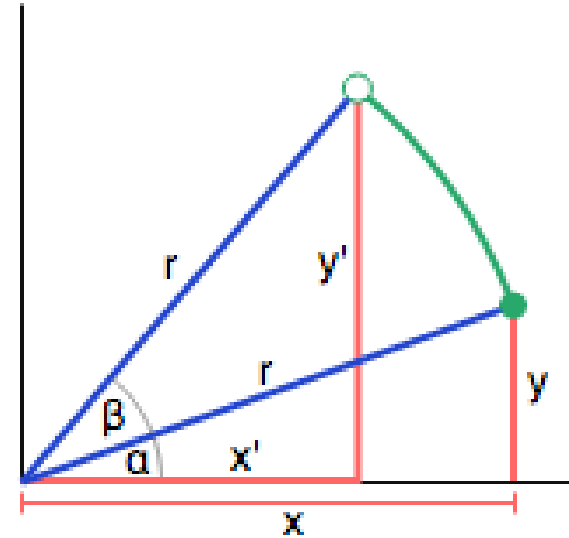


Diagrama de triângulo

Usando algumas identidades trigonométricas, temos:

$$x' = r \times \cos(\alpha) \cos(\beta) - r \times \sin(\alpha) \sin(\beta)$$

$$y' = r \times \sin(\alpha) \cos(\beta) + r \times \cos(\alpha) \sin(\beta)$$

Substituindo os valores de  $x$  e  $y$  acima, temos uma equação para as novas coordenadas como uma função das coordenadas antigas e do ângulo de rotação:

$$x' = x \times \cos(\beta) - y \times \sin(\beta)$$

$$y' = y \times \cos(\beta) + x \times \sin(\beta)$$

Um sistema de equações já pronto a por dentro numa matriz :

$$\begin{Bmatrix} x_2 \\ y_2 \end{Bmatrix} = \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix} \times \begin{Bmatrix} x_1 \\ y_1 \end{Bmatrix}$$

se acrescentarmos o nosso Z para ele não ficar sozinho, a matriz de rotação em torno de Z fica:

$$\begin{Bmatrix} x_2 \\ y_2 \\ z_2 \end{Bmatrix} = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{Bmatrix} x_1 \\ y_1 \\ z_1 \end{Bmatrix}$$

Daí por um raciocínio análogo, conseguimos obter as matrizes de rotação para cada um dos 3 eixos:

**Em torno de Z**

$$\begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

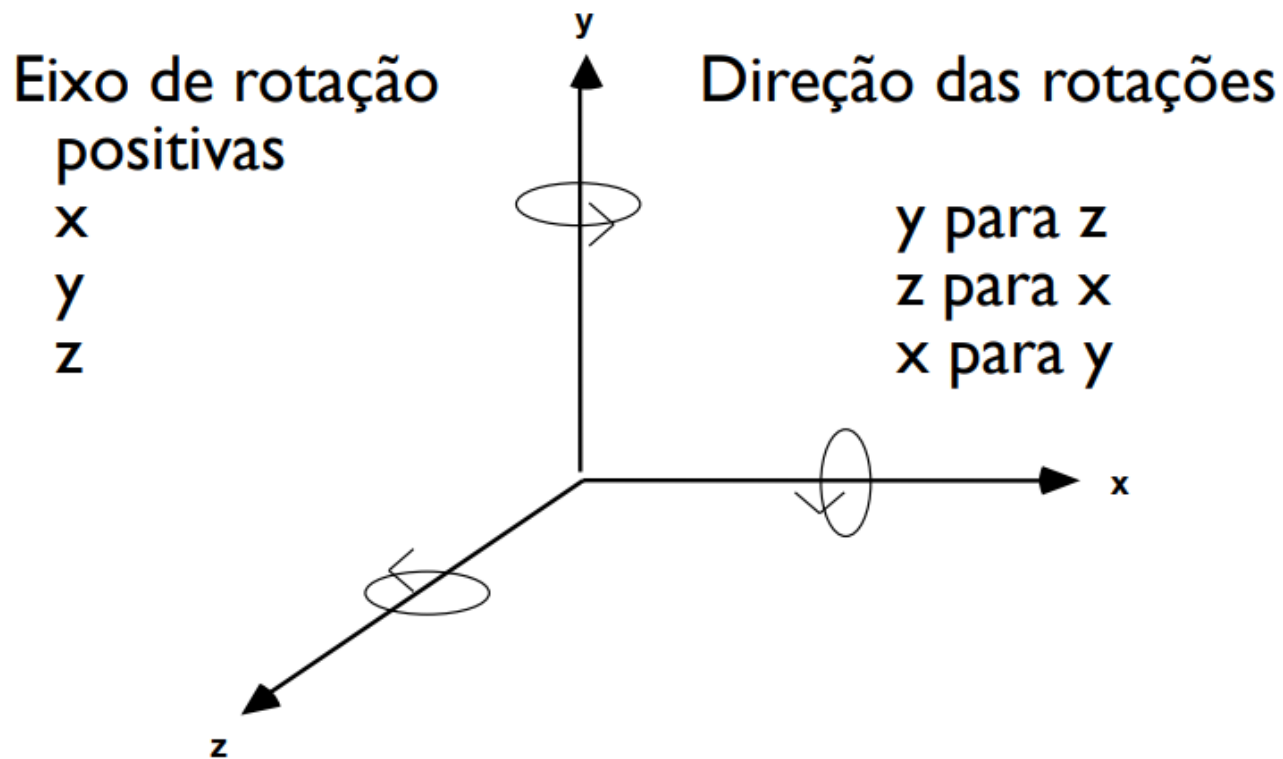
**Em torno de Y**

$$\begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

**Em torno de X**

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) \\ 0 & \sin(\beta) & \cos(\beta) \end{bmatrix}$$

Rotações positivas são definidas como:



As matrizes de rotação de um ângulo  $\theta$  em torno dos eixos  $x$ ,  $y$  e  $z$ , respectivamente, são dadas por  $P_x(\theta)$ ,  $P_y(\theta)$  e  $P_z(\theta)$ :

Uma maneira interessante de projetar pontos do  $R^3$  no plano  $R^2$  é através de duas rotações de ângulos  $\theta$  e  $\phi$  em torno dos eixos  $y$  e  $z$ , respectivamente, seguidas da projeção no plano  $yOz$  definida por

$$P(x, y, z) = (0, y, z).$$

Dessa forma, um ponto  $(x, y, z)$  é projetado no plano  $x = 0$  em

$$(0, x \cos \theta + y \sin \theta, x \sin \theta + y \cos \theta)$$

$$P_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$

$$P_y(\theta) = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix}$$

$$P_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

### Rotação ao redor do **eixo Z**

Forma algébrica:

$$\begin{aligned}x' &= (x * \cos(ang)) + (y * (-\text{sen}(ang))), \\y' &= (x * \text{sen}(ang)) + (y * \cos(ang)), \\z' &= z,\end{aligned}$$

### Rotação ao redor do **eixo X**

Forma algébrica:

$$\begin{aligned}x' &= x, \\y' &= (y * \cos(ang)) + (z * -\text{sen}(ang)), \\z' &= (y * \text{sen}(ang)) + (z * \cos(ang)),\end{aligned}$$

### Rotação ao redor do **eixo Y**

Forma algébrica:

$$\begin{aligned}x' &= (x * \cos(ang)) + (z * -\text{sen}(ang)), \\y' &= y, \\z' &= (x * \text{sen}(ang)) + (z * \cos(ang)),\end{aligned}$$

### Exemplos de programs

- Exemplo 1
- Exemplo 2
- CruzDeCubos

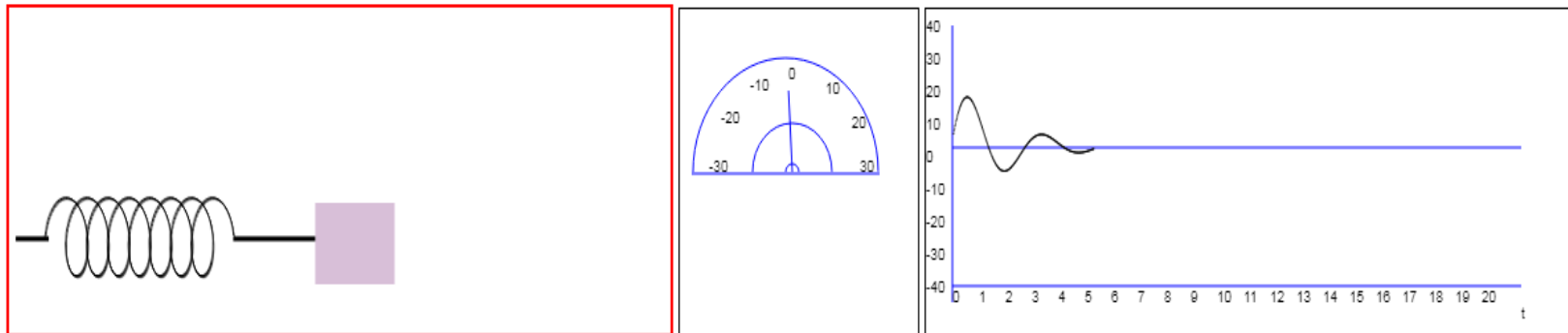
## 4 Programas

### 4.1 Sistema Massa Mola Amortecido

#### Departamento Acadêmico de Física - Ji-Paraná

Coordenador Antonio F. Cardozo - Leis de Newton e Força de Atrito

O oscilador massa-mola é constituído por um corpo de massa  $m$  ligado a uma mola de constante elástica  $k$ , presa a uma parede (verticalmente ou horizontalmente). Cada mola tem a sua constante elástica, que depende do material de que é feita e da sua geometria. O corpo executa o MHS sobre uma superfície horizontal sem atrito. Veja a Figura 1. Quando a mola é comprimida (ou esticada) e liberada, o corpo passa a executar um movimento unidimensional de vai-e-vem. O movimento é regido pela Lei de Hooke, que relaciona a força restauradora com o deslocamento da massa:  $F=-kx$ .



Executat Parar Limpar Amplitude  Massa(kg)\*   
k (N/m)\*\*  R\*\*\*\*\*

Y(m) t(s)

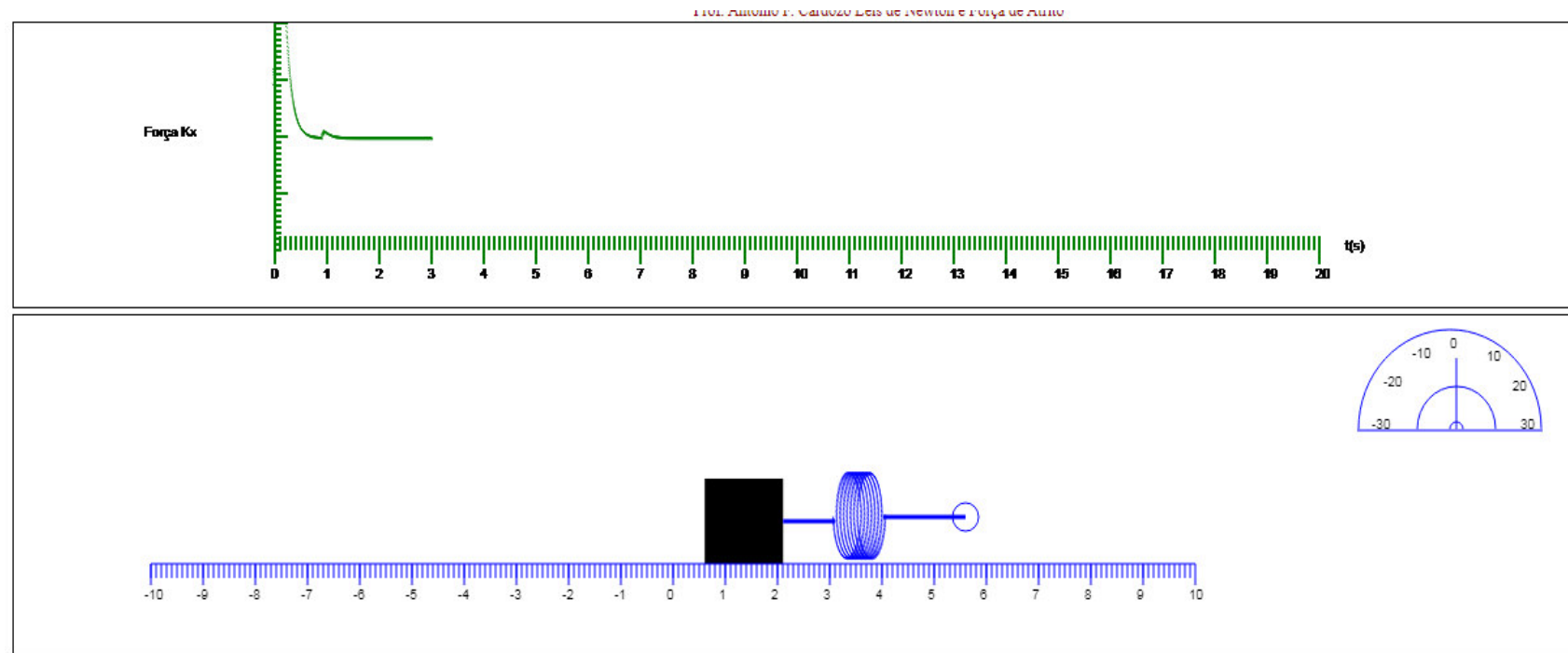
-1.3894	20.20
-1.1187	20.30
-0.8510	20.40
-0.5869	20.50
-0.3268	20.60
-0.0711	20.70

## 4.2 Lançamento e Queda Livre quedal.Jjava





## 4.3 Força de atrito e lei de hooke



Calcular Parar Limpar teste

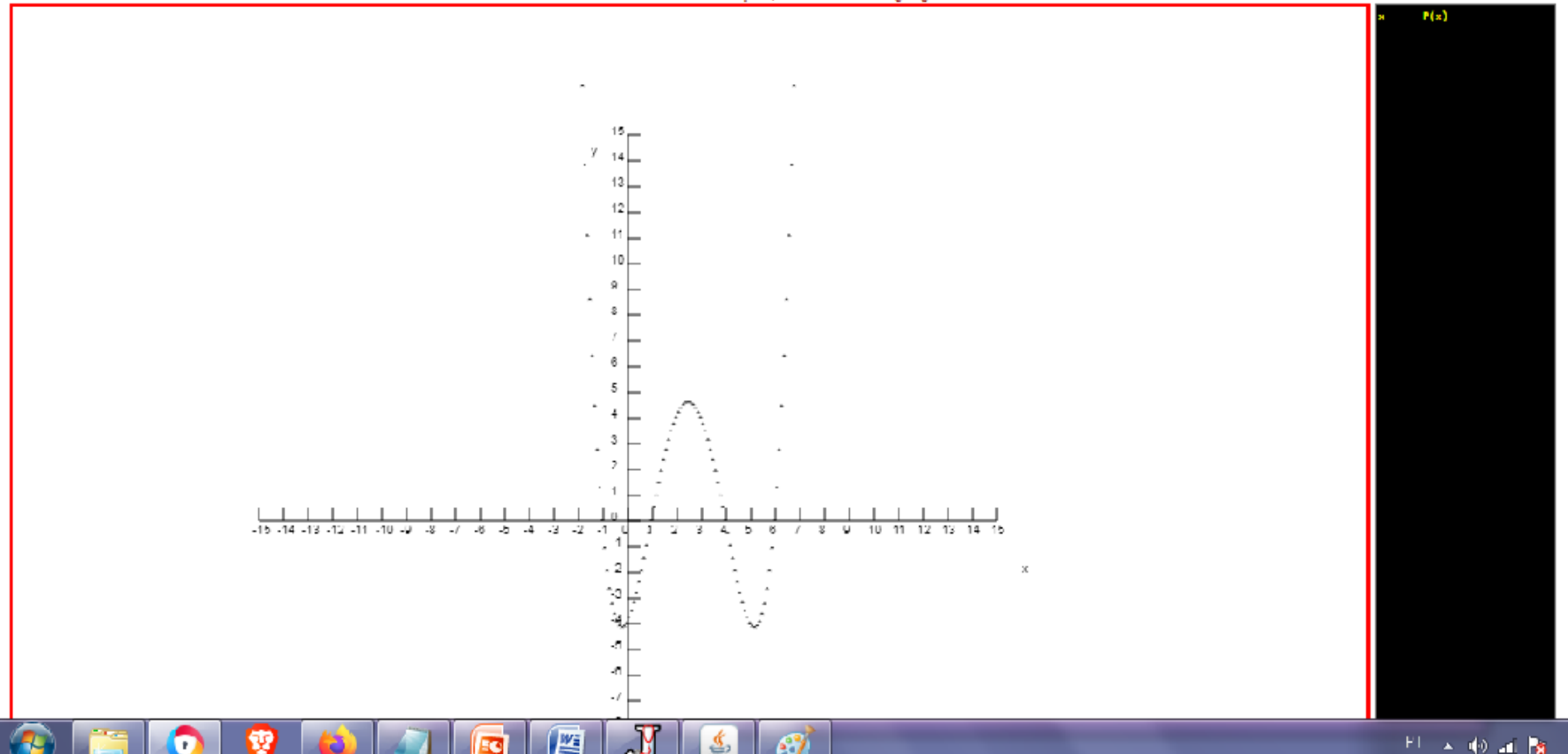
Massa 1 C Est 0 C Din 0.1 K 12

Delta x 0 tempo 609.06 Px 1 Pmause 729

## 4.4 Interpolacao Numerica

Departamento Acadêmico de Física - Ji-Paraná

Prof. Antonio F. Cardozo  
Interpolação Polinomial de Lagrange



## 4.5 Movimento de Partículas

FORÇA DE ATRITO.html

Partículas H65 84/10/592 (3).htm

Departamento Acadêmico de Física - Ji-Paraná

Prof. Antonio T. Cardozo

Movimento de Partículas em trajetórias forçadas (Imagem e Keypres)

INICIAR Parar

Massa\*\*\* 1 Energia\*\* 1 Atracão\* 0

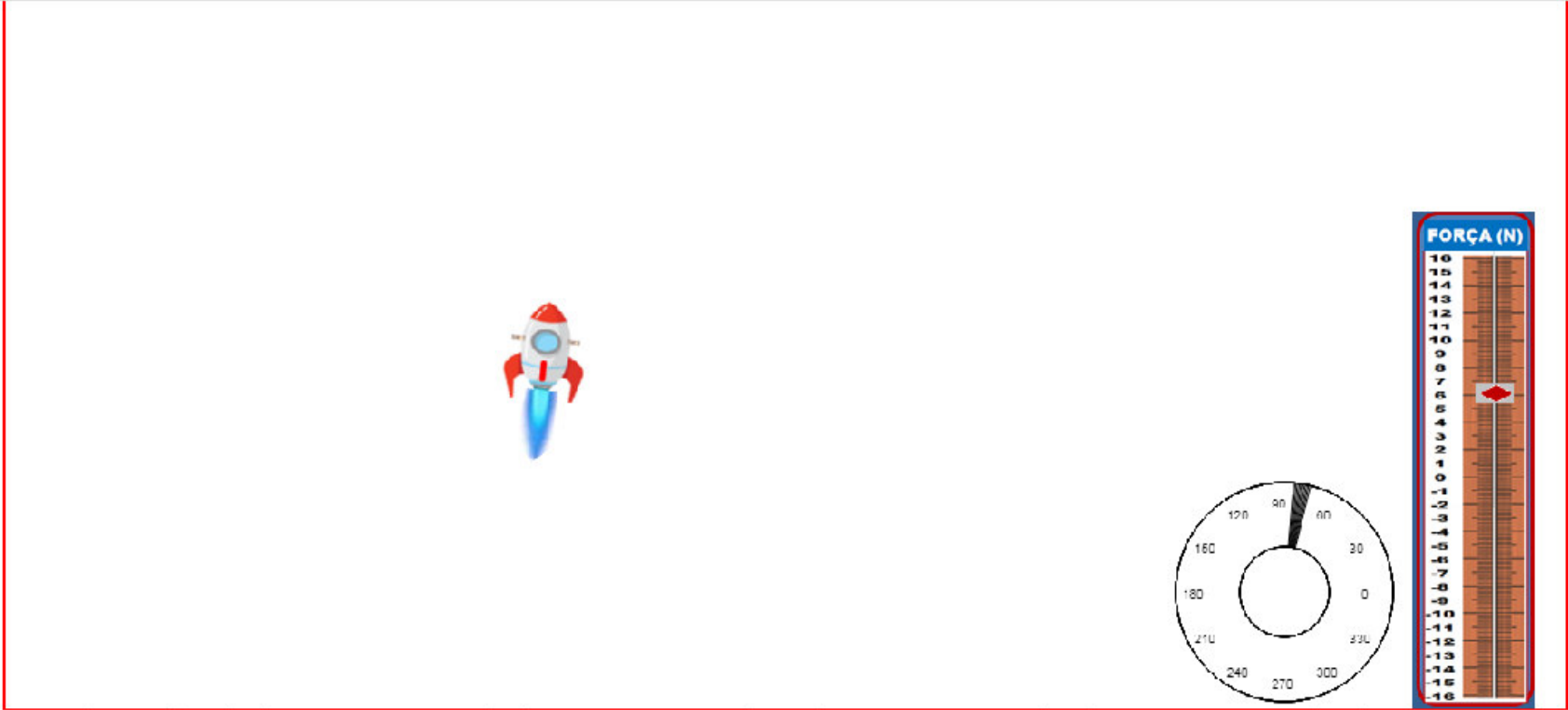
21:07 26/11/2023

## 4.6 Movimento Uniforme e Acelerado

FORÇA DE ATRITO.html x Movimento Uniforme e Acelerado x

Arquivo | D:/DEPARTAMENTO/MINICURSO%202023/curso/AMOSTRAS%20HTML/Movimento%20Uniforme%20e%20Acelerado.html

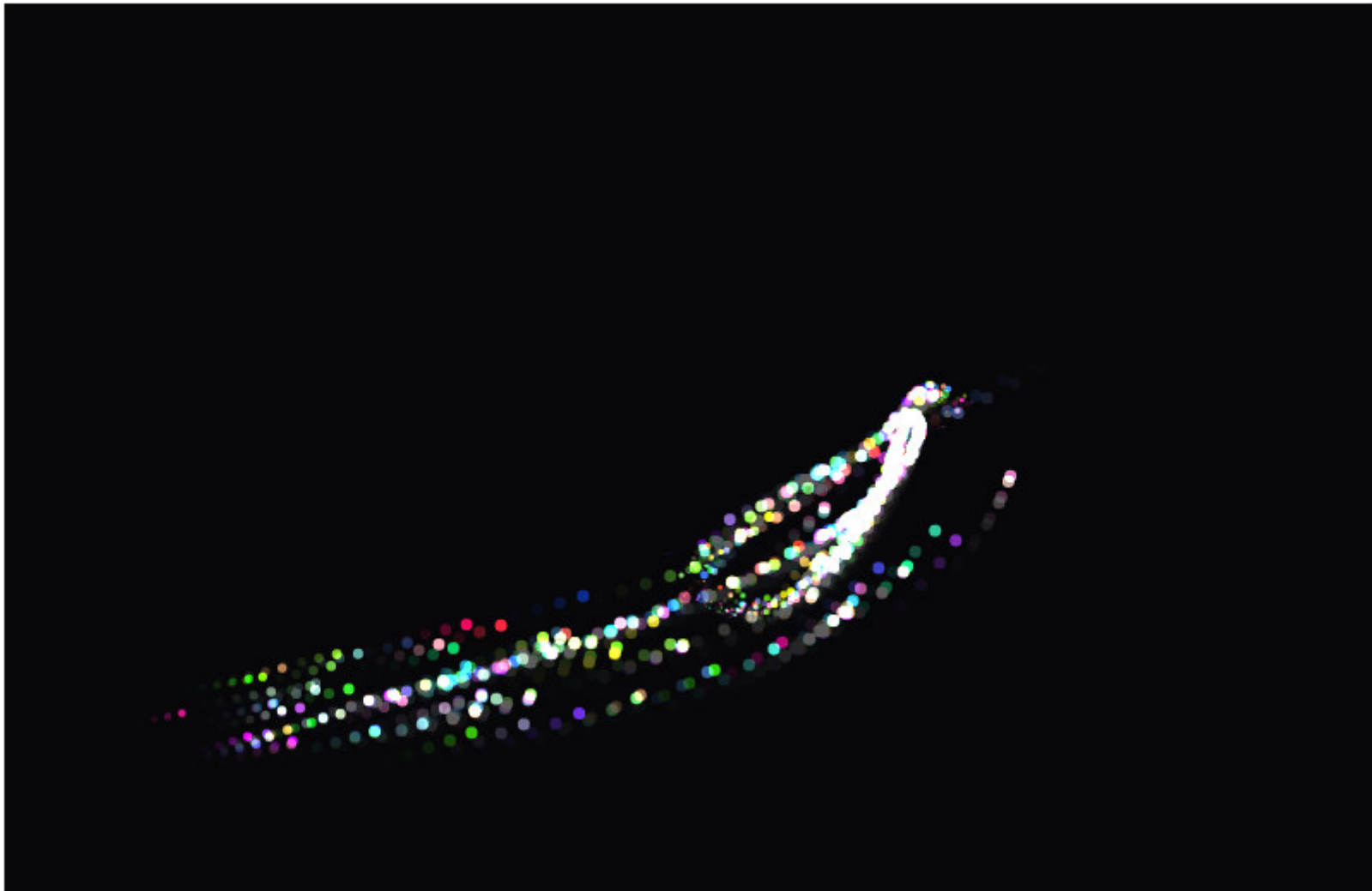
Google Address Store AliExpress Facebook YouTube Booking.com Google



The simulation interface features a central rocket icon, a circular gauge with a needle pointing to approximately 60 degrees, and a vertical force scale labeled "FORÇA (N)" ranging from -16 to 10. The force scale has a red diamond marker at approximately 5.5 N.

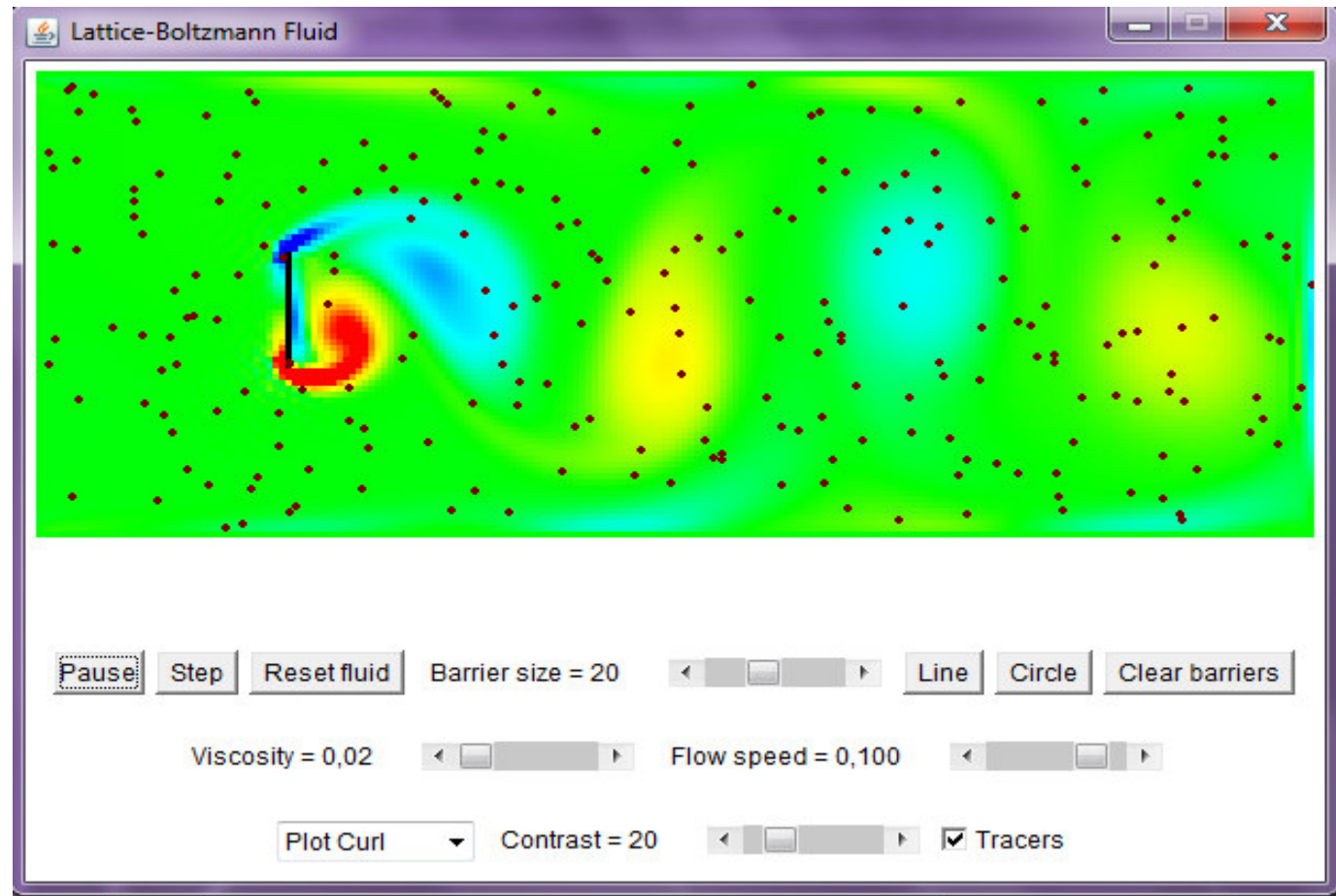
Calculador PARAR teste X0 519 490 Y0 1 Massa 10 V0x 1 V0y 1

## Movimento de Partículas 2



## 4.7 Demonstração de treliça Boltzmann

[Java](#)   [Html](#)



- 1. <https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent/button>
- 2. <https://desenvolvimentoparaweb.com/javascript/this-javascript-dominando/>
- 3. <https://www.manualcodigo.com.br/curso-javascript-p5-13/>
- HALLIDAY David; RESNICK Robert; Jearl Walker Fundamentos de Fisica, 1-Mecanica, 4E
- MAIER RV Problemas, algoritmos, software: e-livro. - Olhos: Glazovskaya. Estado. ped. Inst, 2012.  
URL: <http://maier-rv.glazov.net> Ultima consulta 15/09/2015