DEPARTAMENTO DE FÍSICA

Simulação de Fenômenos Físicos em JavaScript

Prof. Antonio F. Cardozo

⓿Documento HTML

❶JavaScript

❷HTML5 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

⓿ Documento HTML

0.2 Tabelas

0.3 Formulários

❶JavaScript

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

1.5 Recursos para interação com os 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 Funão timedCount()
1.5.7 Saída de dados com alert ()
1.5.8 Entrada de dados com prompt()

❷HTML5 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

❸Codificação 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

❹ Programas

4.1 Sistema Massa Mola Amortecido

⓿ 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;

** ** – 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:
** ** – Principal tag de texto, compõe um parágrafo;
**** – 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;
**** – Transforma o conteúdo em negrito;
**** – Transforma o conteúdo em itálico;
**
** – 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>** com o atributo href. Por exemplo, caso você queira criar um link no seu texto que redirecione à página inicial do google:
Para acessar o Google, clique aqui.

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:

<img src="imagens/imagem1.jpg" alt="imagem 1 da minha página">

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

https://www.homehost.com.br/blog/tutoriais/tags-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/

❶JavaScript

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

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

1.2.2 Operadores de Comparação

Comparação

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

1.2.3 Operadores Bit a bit

4.2.3 Bit a bit

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

1.2.4 Operadores de Atribuição

Atribuição

Operador	Exemplo	Equivalente
=	x = 2	Não possui
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
&=	x &= y	x = x & y
|=	x |= y	x = x | y
^=	x ^= y	x = x ^ y
>>=	x >>= y	x = x >>= y
<<=	x <<= y	x = x <<= y
>>>=	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) * 57.3248000000000003);
        form.campo22.value=Ang0.toFixed(3);
    if(ht < 0.0)
        Ang0 = 180 + Math.abs(Math.atan(ht / xt) * 57.3248000000000003);
        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.

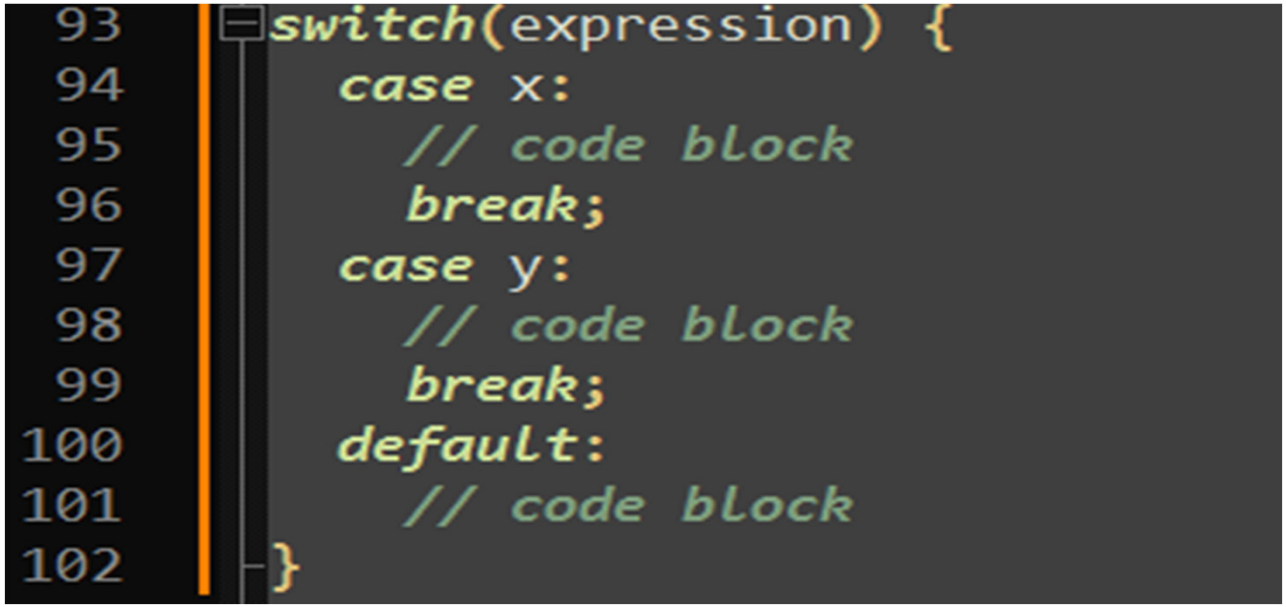
Também é possível aglomerar mais testes, utilizando-se o comando *else if*.

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

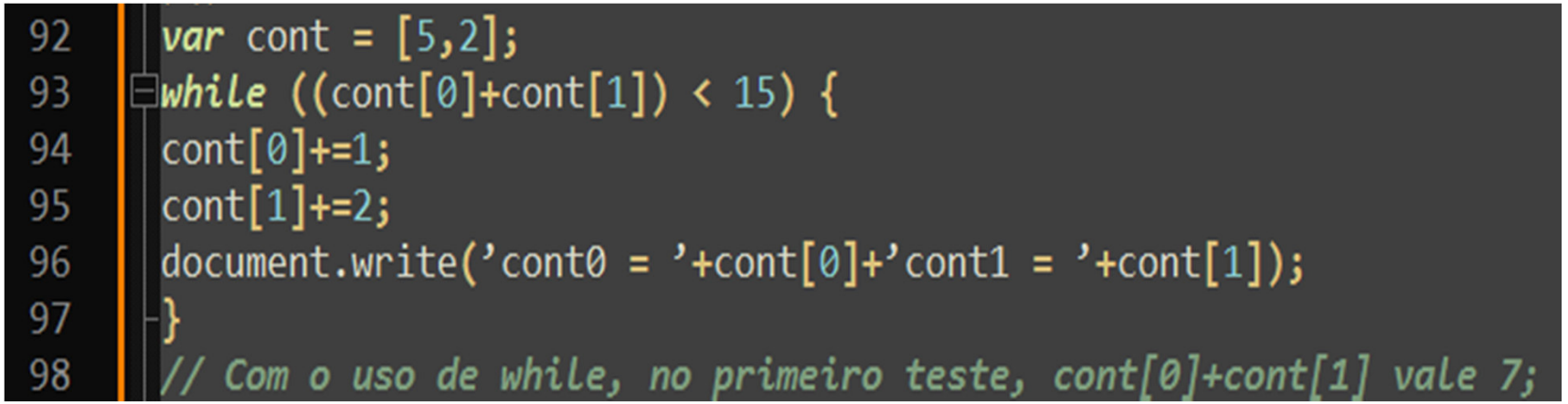
As estruturas do tipo switch são usadas quando queremos selecionar uma opção dentre várias disponíveis.



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:

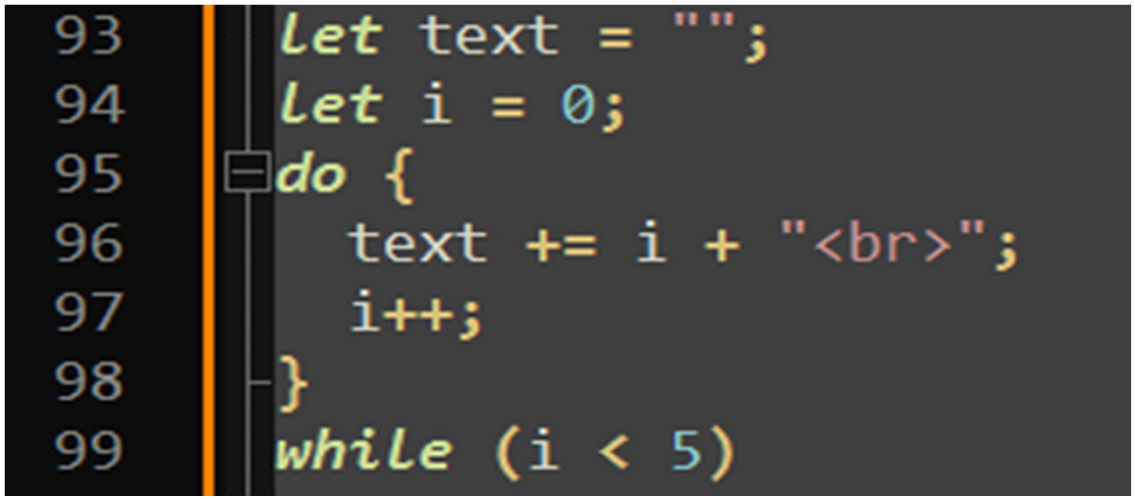


1.2.9 do ... while

Diferentemente do while, o do ... while primeiro executa o conteúdo do laço uma vez e, depois disso, realiza o teste da expressão pare 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:



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']]
```

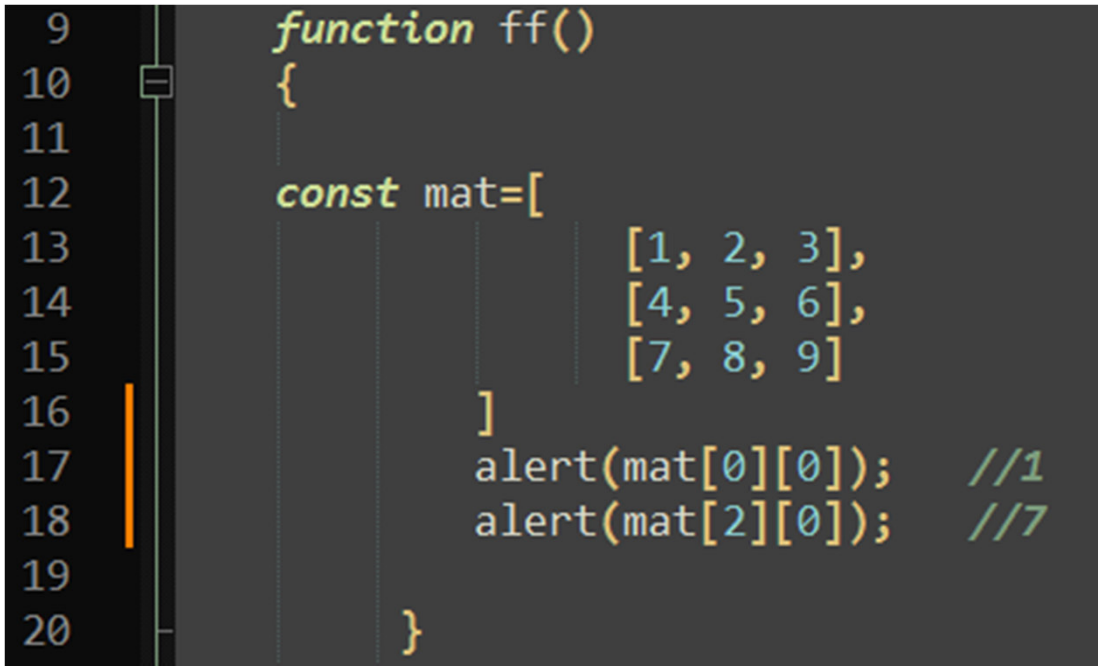
Como podemos incluir um *array* dentro de um *array*, podemos criar *arrays* multidimensionais, que possuem aplicações muito úteis (por exemplo, uma matriz):

```
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.



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

```
function ff()
{
form.textarea.value += "m   "+ "v   "+'\r\n';

    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],
                   [0,0,0,0,0,0,0,0,0,0,0],[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],
                   [0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0],];
    for(i=0;i<11;i++)
    {
       for (p=0;p<11;p++)
       {
          Matrix_A[i][p]=i+p;
       }
     }
    for(i=0;i<11;i++)
    {
       for (p=0;p<11;p++)
       {
       form.textarea.value += "[i][p]=  "+ Matrix_A[i][p]+'\r\n';
       }
     }
    }
```

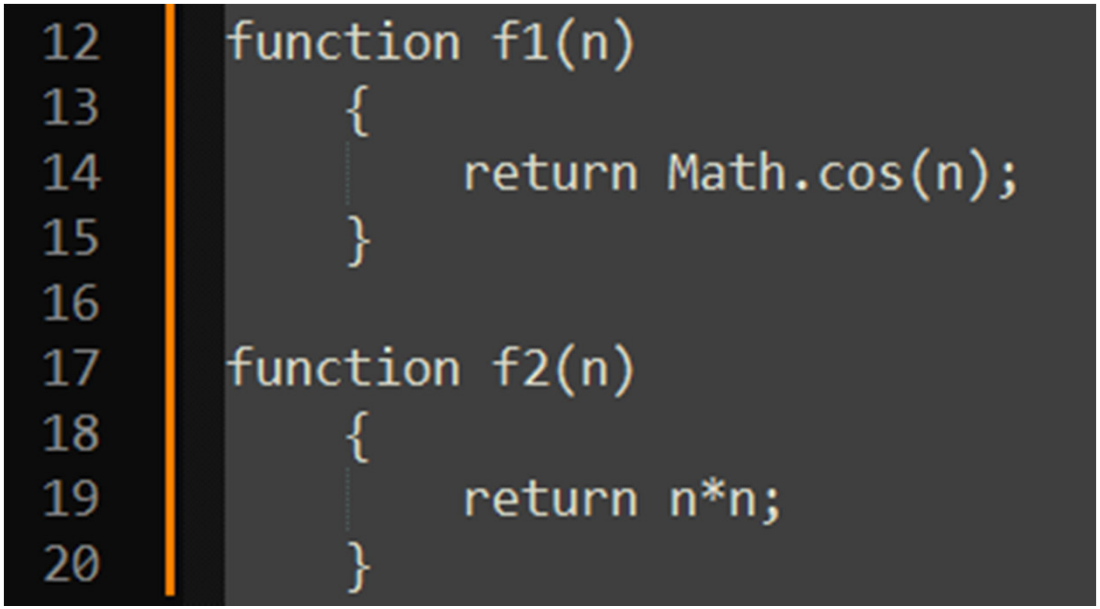
1.3 Funções

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 { }.



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

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

1.4 Objetos em Javascript

CommandBu

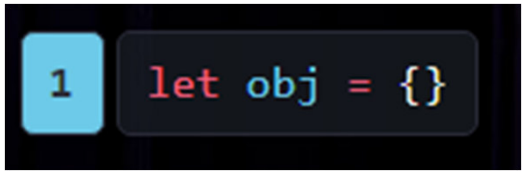
Introdução

Objeto em Javascript é uma estrutura na qual você pode juntar variás 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:



1.4.1 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 vírgulos, 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 variáves relacionadas em um lugar só, acessando-as através do seu objeto. Isso é conceito de **orientação a objetos**, onde criarmos 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:

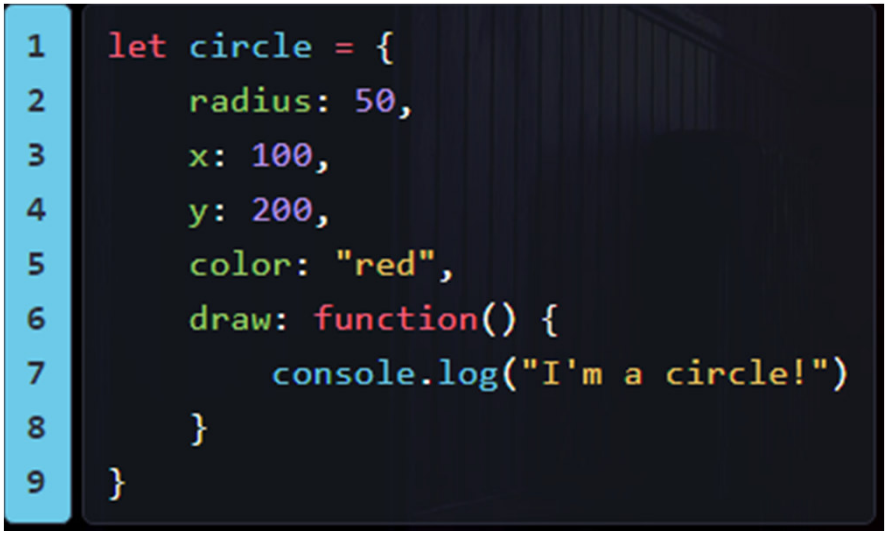
```
1  console.log(circle["radius"])
```

brackets notation

https://www.manualdocodigo.com.br/curso-javascript-p5-13/#:~:text=Objeto%20em%20Javascript%20%C3%A9%20uma,de%20um%20objeto%20de%20propriedades.

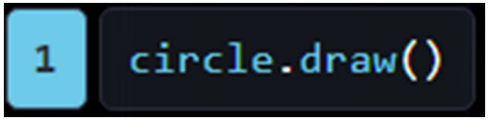
1.4.2 Métodos em um objeto

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:



```
let circle = {
    radius: 50,
    x: 100,
    y: 200,
    color: "red",
    draw: function() {
        console.log("I'm a circle!")
    }
}
```

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



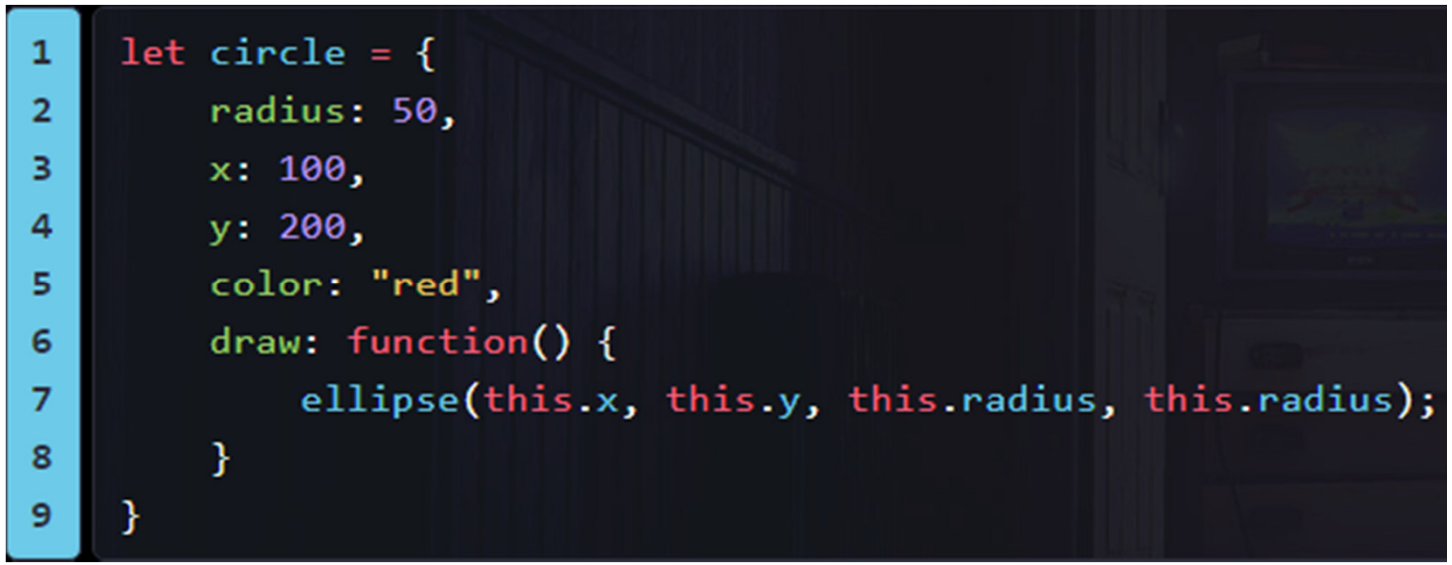
```
circle.draw()
```

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 preopriedades 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:



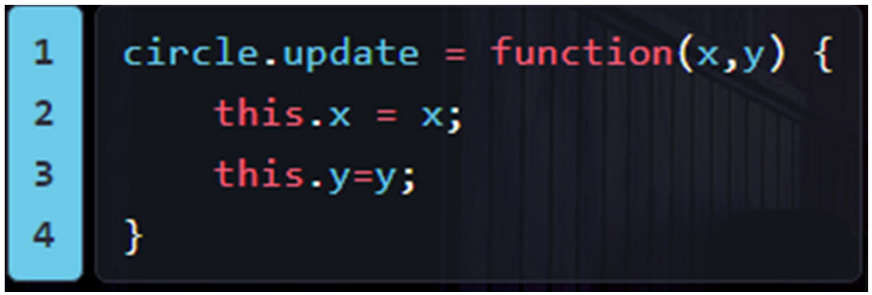
```
let circle = {
    radius: 50,
    x: 100,
    y: 200,
    color: "red",
    draw: function() {
        ellipse(this.x, this.y, this.radius, this.radius);
    }
}
```

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.

```
var person = {
    firstName    : "Penelope",
    lastName     : "Barrymore",
    // Já que a palavra-chave "this" é usada dentro do método showFullName() abaixo e
    // é definido no objeto "person", "this" terá o valor do objeto "person" porque e
    // invocará showFullName().
    showFullName : function() {
        console.log ( this.firstName + ' ' + this.lastName );
    }
}

person.showFullName(); // Penelope Barrymore
```

Exemplo em jQuery

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

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. DblClick
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

```
 <! DOCTYPE html>
<html lang = "kr" >
    <head><meta charset = "utf-8" > </head>
    <body>
        <canvas id = "tcanvas" width = "1000" height = "200" style =' border : 1px solid #000 '; > </canvas>
        <script>
            var canvas = document. getElementById ( "tcanvas" );
            var context = canvas. getContext ( "2d" );

            canvas. addEventListener ( "mousedown" , function (me) {mDown (me)}, false );
            canvas. addEventListener ( "mousemove" , function (me) {mMove (me)}, false );
            canvas. addEventListener ( "mouseup" , function (me) {mUp (me)}, false );
            canvas. addEventListener ( "mouseout" , function (me) {mOut (me)}, false );
            var stX =0;
            var stY =0;
            var drag = false ;
            function mDown(me)
            {
                stX = me. offsetX ;
                stY = me. offsetY ;
                form.campo21.value=me.button;
                drag = true ;
            }
```

MausUp

MauseOp

```
<! DOCTYPE html>
<html lang = "kr" >
    <head><meta charset = "utf-8" > </head>
    <body>
        <canvas id = "tcanvas" width = "1000" height = "200" style =' border : 1px solid #000 '; > </canvas>
        <script>
            var canvas = document. getElementById ( "tcanvas" );var canvas = document. getElementById ( "tcanvas" );
            canvas. addEventListener ( "mousedown" , function (me) {mDown (me)}, false );
            canvas. addEventListener ( "mousemove" , function (me) {mMove (me)}, false );
            canvas. addEventListener ( "mouseup" , function (me) {mUp (me)}, false );
            canvas. addEventListener ( "mouseout" , function (me) {mOut (me)}, false );
            var stX =0; var stY =0;
            var drag = false ;
            function mDown(me)
            {
                stX = me. offsetX ; stY = me. offsetY ;
                form.campo21.value=me.button;drag = true ;
            }
            function mMove(me)
            {
                if (!drag)
            {
                return ;
            }
            var nowX = me. offsetX ;var nowY = me. offsetY ;canvasDraw (nowX,nowY);
            stX = nowX; stY = nowY;
            form.campo22.value="move";
            }
```

```
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 while loop	let i = 0 while (i < 3)
	{
	console.log(i); i += 1;
	}

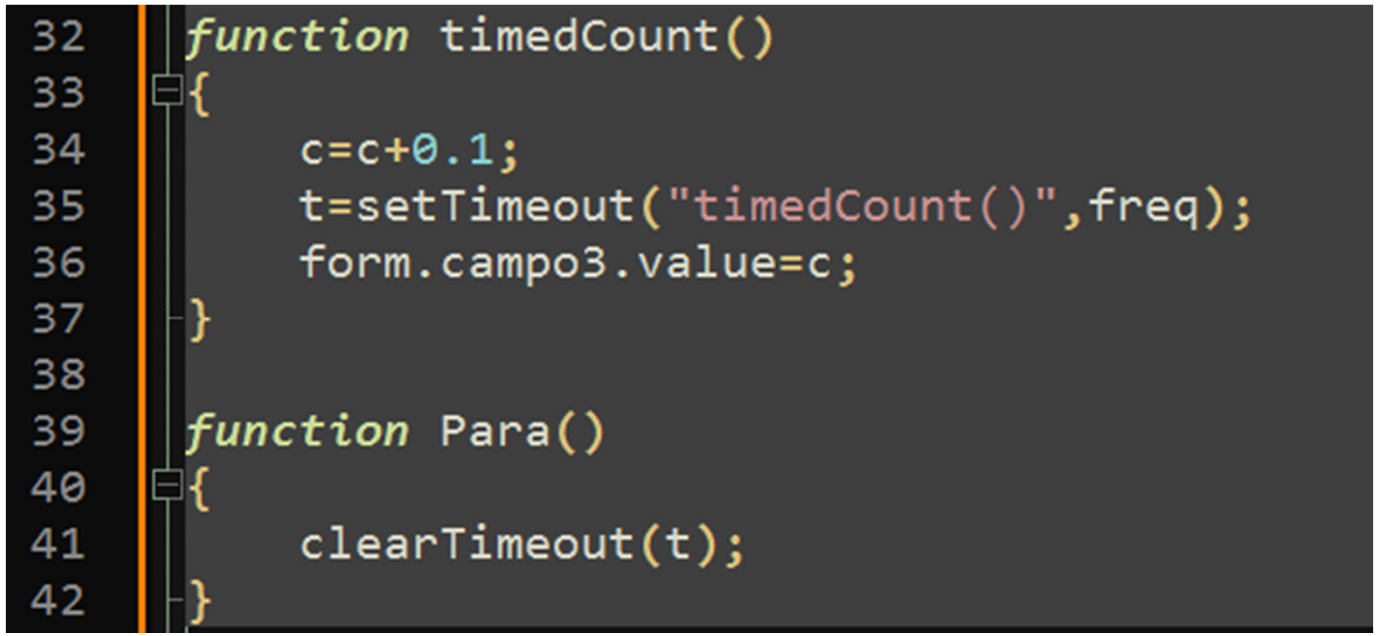
Infinite Loop Using for	for (let i = 0; i < 3; i++)
	{
	console.log(i)
	}

function timedCount()	function timedCount()
	{
	t=setTimeout("timedCount()",freq);
	}

1.5.6 Funão timedCount()

Codigo

Exemplo de TimeCount



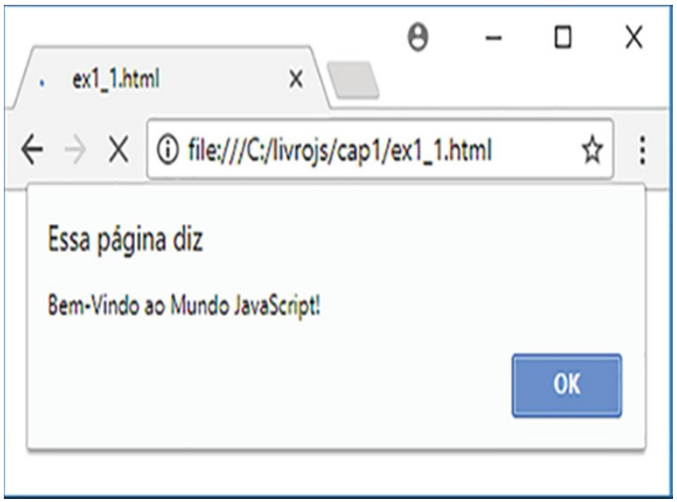
```
32  function timedCount()
33  {
34      c=c+0.1;
35      t=setTimeout("timedCount()",freq);
36      form.campo3.value=c;
37  }
38
39  function Para()
40  {
41      clearTimeout(t);
42  }
```

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>
```

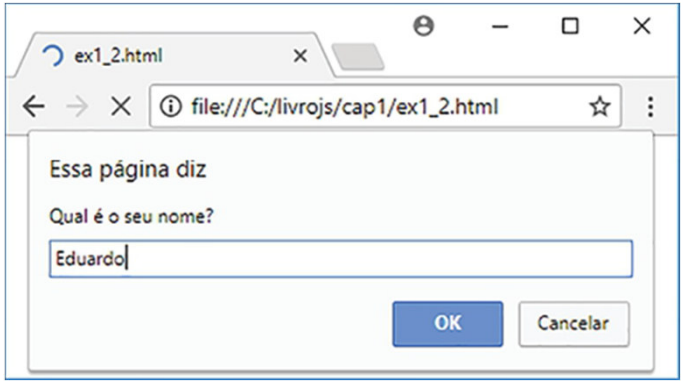


1.5.8 Entrada de dados com prompt()

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>
```



❷HTML5 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>
		</head>
</html>
```

2.3 O sistema de coordenadas do Canvas

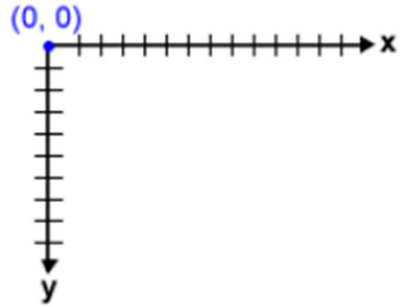
Para posicionarmos os desenhos no Canvas, pensamos nele como um enorme conjunto de pontos. Cada ponto possui uma posição horizontal (x) e uma vertical (y). O ponto (0, 0) (lê-se: zero em x e zero em y) corresponde ao canto superior esquerdo do Canvas:

```
<!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

```
<canvas id="meu_canvas" width="200" height="200"></canvas>
<script>
// Canvas e contexto
var canvas = document.getElementById('meu_canvas');
var context = canvas.getContext('2d');

// Desenhar um retângulo
context.fillRect(50, 50, 100, 100);
</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

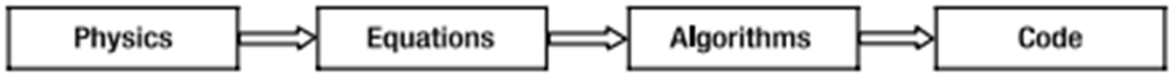
Esses comandos não desenham as linhas imediatamente, apenas armazenam as informações no path. Devemos chamar o método stroke para desenhá-las de fato através do moveTo e lineTo.

```
<canvas id="meu_canvas" width="300" height="300"></canvas>
<script>
    // Canvas e contexto
    var canvas = document.getElementById('meu_canvas');
    var context = canvas.getContext('2d');
    // Iniciar o caminho (apaga desenhos anteriores)
    context.beginPath();
    // Desenhar uma estrela
    context.moveTo(75, 250); // Ponto inicial
    context.lineTo(150, 50);
    context.lineTo(225, 250);
    context.lineTo(50, 120);
    context.lineTo(250, 120);
    context.lineTo(75, 250);
    // Configurar a linha
    context.lineWidth = 2;
    context.strokeStyle = 'red';
    // Traçar as linhas do caminho
    context.stroke();
</script>
```

3. Codificação de problemas físicos

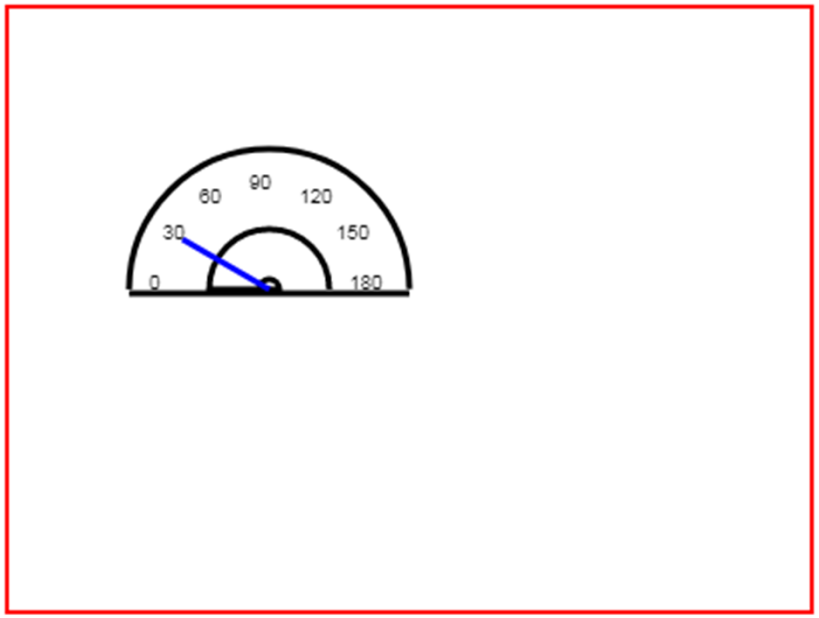
Depois de conhecer um pouco de física básica (e um pouco de matemática relevante), codificá-la não será muito diferente ou mais difícildo que você está acostumado como programador, desde que faça isso da maneira certa. O caminho certo é descrever o que está envolvido na simulação da física real e como 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 serdividido em quatro etapas, conforme mostrado esquematicamente na Figura 1-1.



3.1 Exemplo Indicador de Graus

Código



desenhar

Angulo 30

3.2 Exemplo Time Pressão

Código

Minicurso JavaScript

Executat Parar Limpar

Freq 100 36

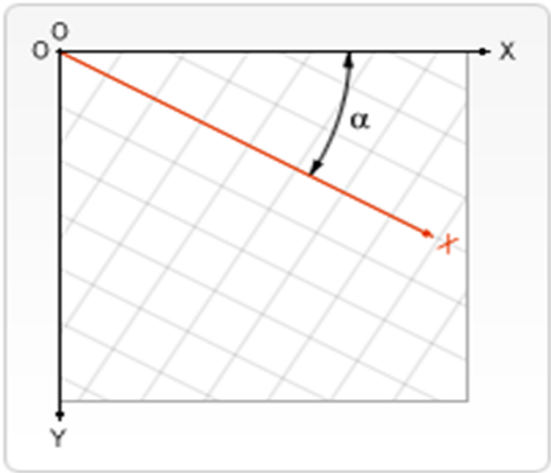
Rotação

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.

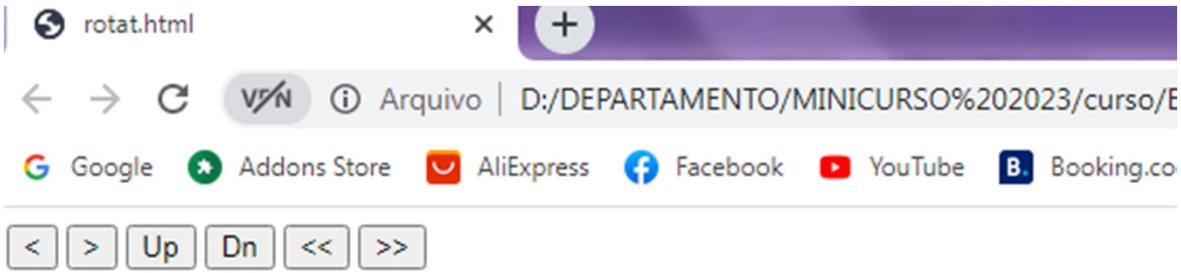


O ponto central de rotação sempre é o ponto de origem. Para alterar o ponto central, precisamos mover o canvas usando o método translate().

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 **α**, então

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

Se girarmos mais β até o ponto (x', y'), então:

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

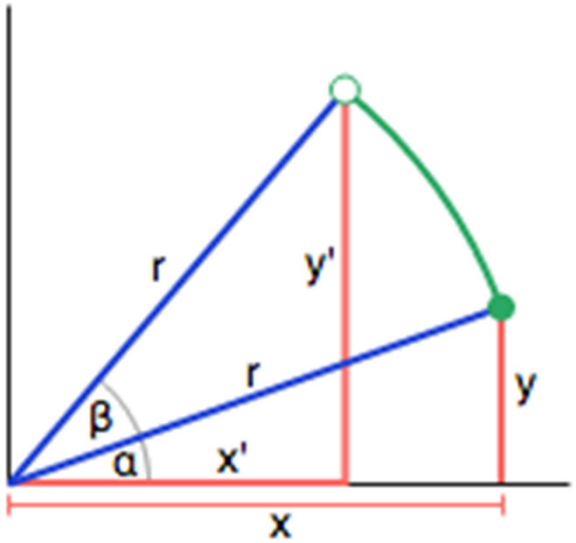


Diagrama de triângulo

Usando algumas identidades trigonométricas, temos:

$x' = r \times \cos(\alpha)\ \cos(\beta) - r \times \text{sen}(\alpha)\ \text{sen}(\beta)$

$y' = r \times \text{sen}(\alpha)\ \cos(\beta) + r \times \cos(\alpha)\ \text{sen}(\beta)$

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:

$x' = x \times \cos(\beta) - y \times \text{sen}(\beta)$

$y' = y \times \cos(\beta) + x \times \text{sen}(\beta)$

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

$$\begin{Bmatrix} X2 \\ y2 \end{Bmatrix} = \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix} x \begin{Bmatrix} x1 \\ y1 \end{Bmatrix}$$

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

$$\begin{Bmatrix} X2 \\ y2 \\ z2 \end{Bmatrix} = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} X \begin{Bmatrix} x1 \\ y1 \\ z1 \end{Bmatrix}$$

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

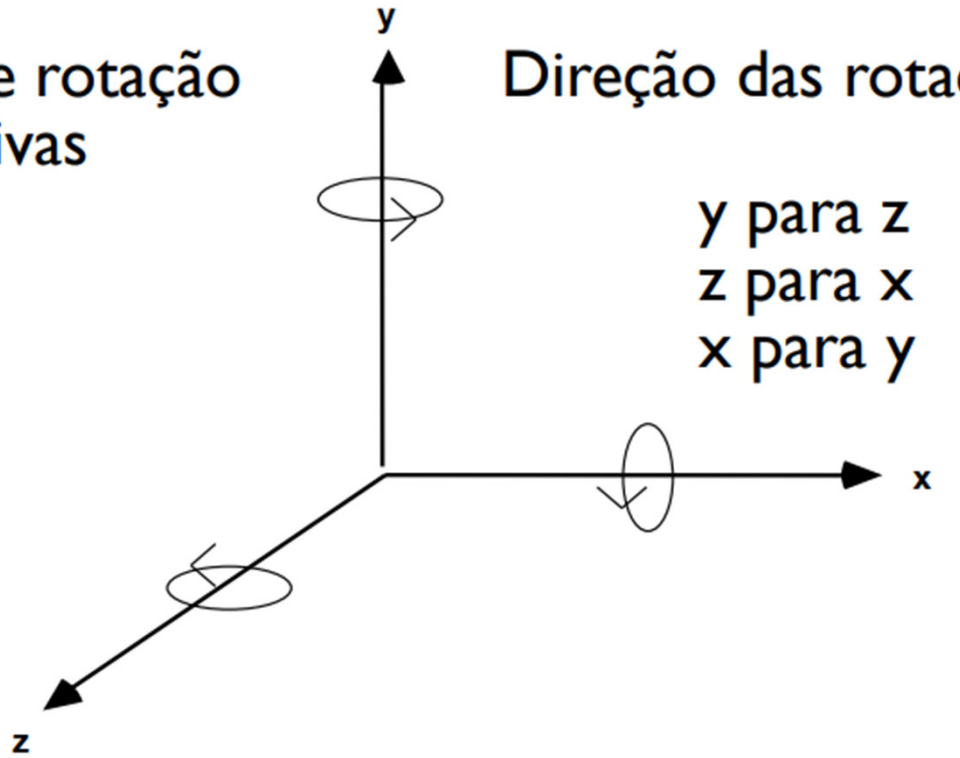
$$\begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

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:

Eixo de rotação positivas	Direção das rotações
x	y para z
y	z para x
z	x para y



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

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

Uma maneira interessante de projetar pontos do R^3 no plano R^2 é através de duas rotações de ângulos θ e ϕ em torno dos eixos y e z, respectivamente, seguidas da projeção no plano y0z definida por

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

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

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

$(0, x \operatorname{sen}\theta + y\cos\theta, \; x\cos\theta \operatorname{sen}\phi - y \operatorname{sen}\theta \operatorname{sen}\phi + z\cos\phi)$

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

Rotação ao redor do **eixo Z**

Forma algébrica:

$$x' = (x * \cos(ang)) + (y * (-sen(ang)))$$
$$y' = (x * sen(ang)) + (y * \cos(ang))$$
$$z' = z;$$

Rotação ao redor do **eixo X**

Forma algébrica:

$$x' = x;$$
$$y' = (y * \cos(ang)) + (z * -sen(ang))$$
$$z' = (y * sen(ang)) + (z * \cos(ang))$$

Rotação ao redor do **eixo Y**

Forma algébrica:

$$x' = (x * \cos(ang)) + (z * -sen(ang))$$
$$y' = y;$$
$$z' = (x * sen(ang)) + (z * -\cos(ang))$$

Exemplos de programs

- Exemplo 1
- Exemplo 2
- CruzDeCubos

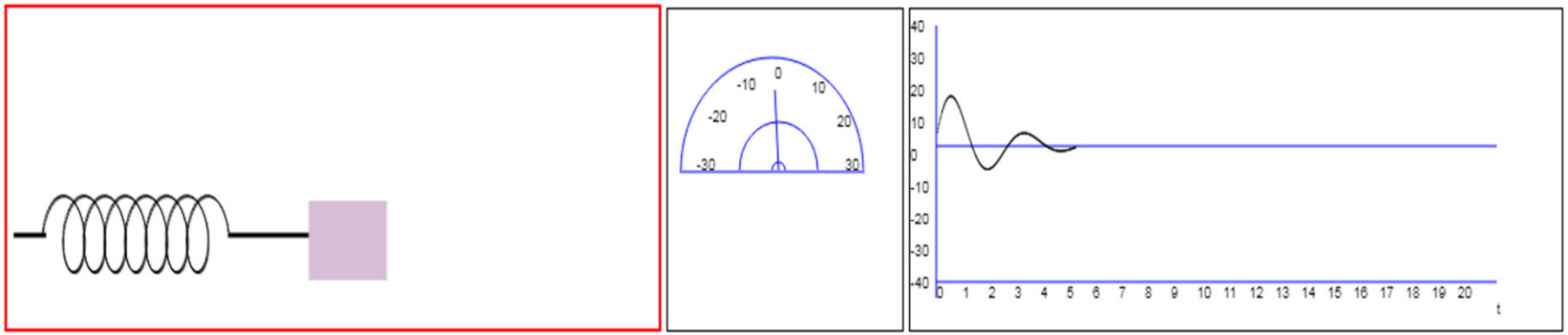
❹ Programas

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.



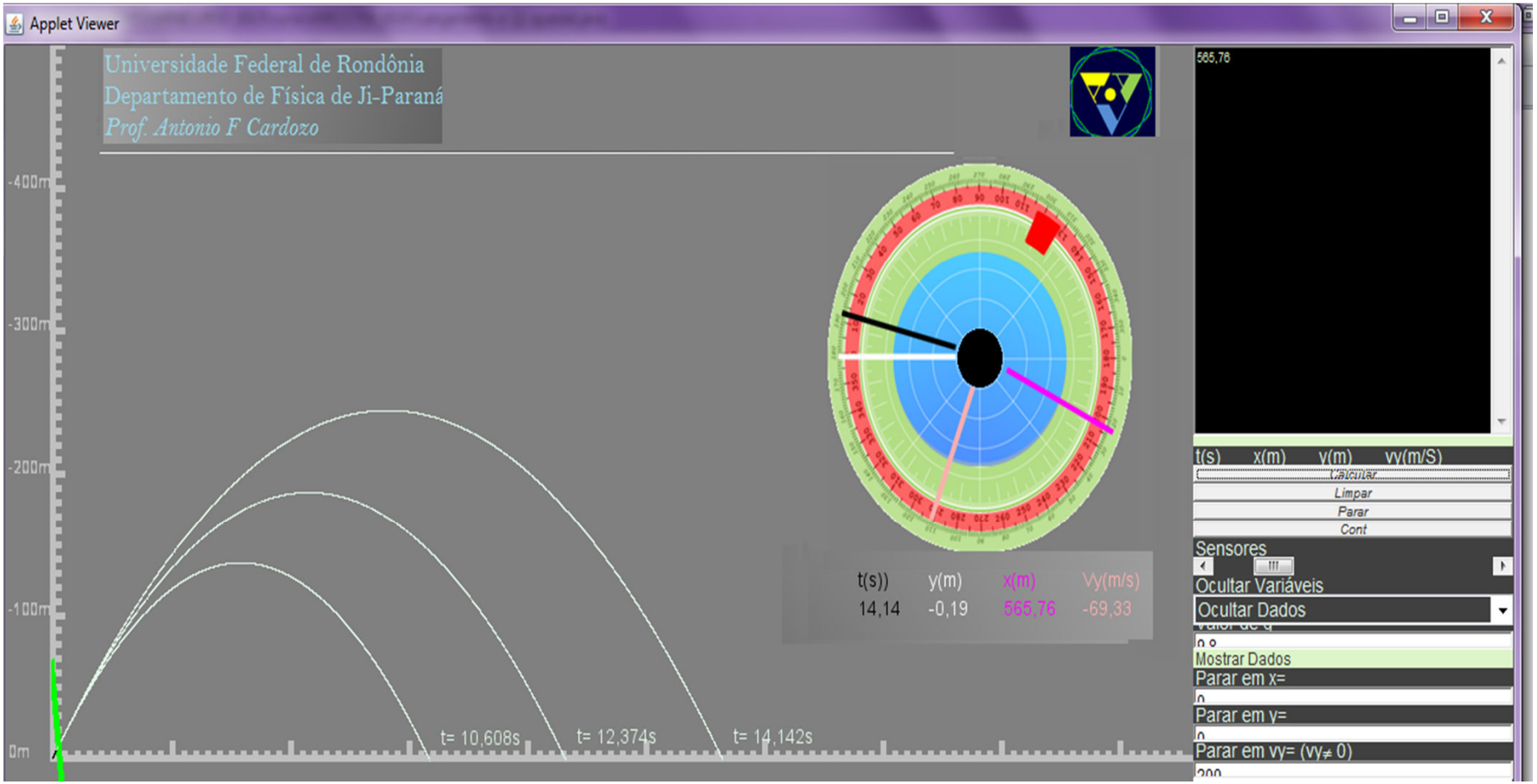
Executat | Parar | Limpar | Amplitude | 20 | Massa(kg)* | 10

k (N/m)** | 2 | R******** | 1 | 0

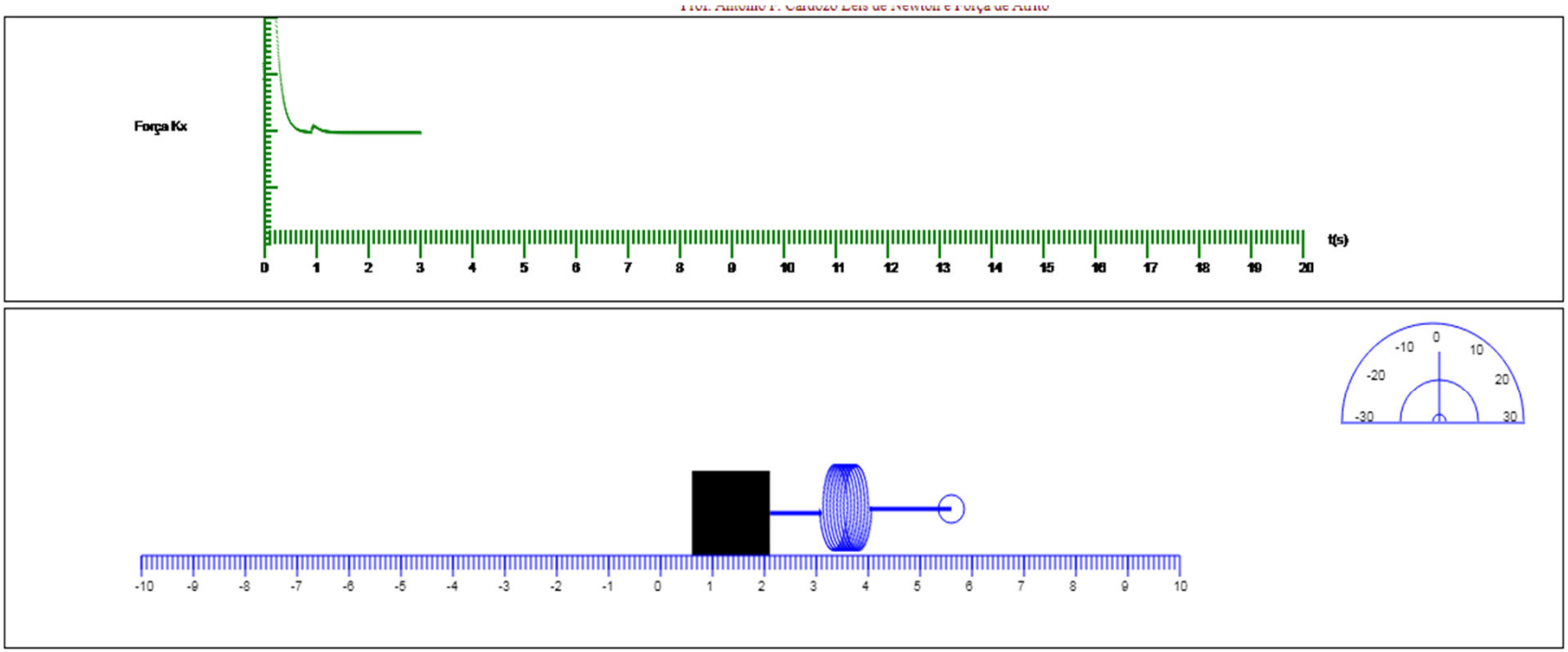
Y(m)_____t(s)

```
-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 Lançameto e Queda Livre quedal.Jjava



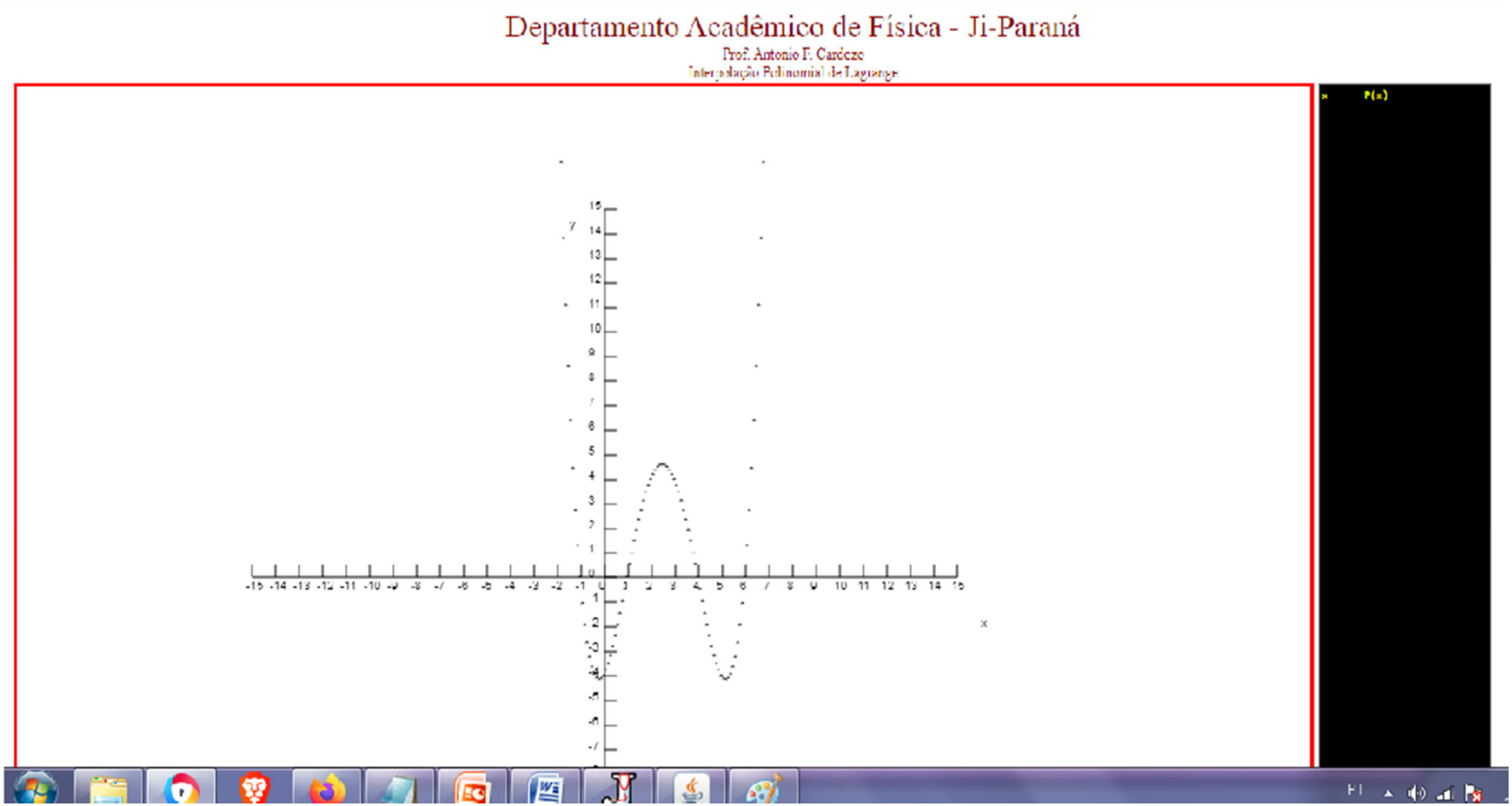
4.3 Forã de atrito e lei de hooke



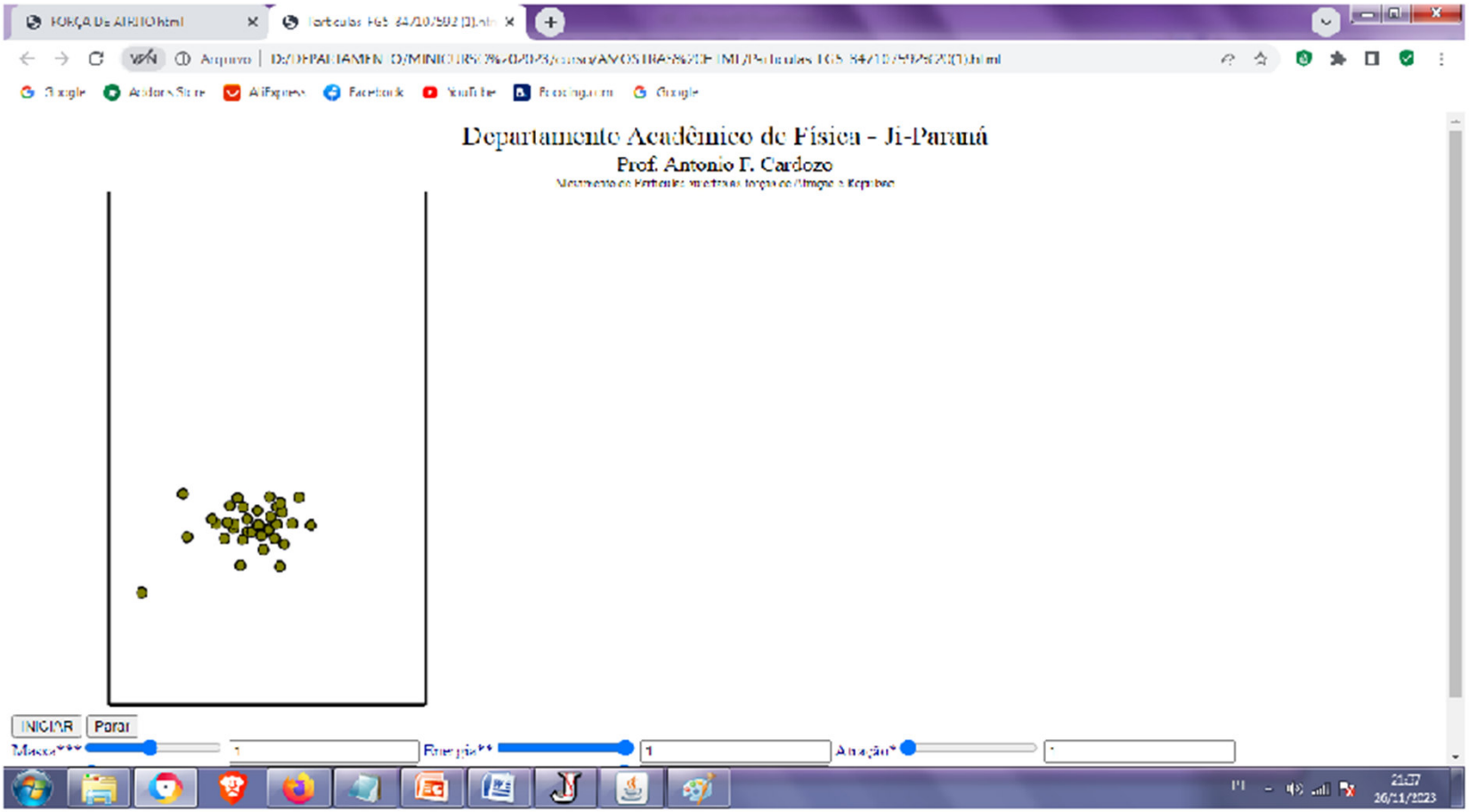
Calcular | Parar | Limpar | teste

Massa	1	C Est	0	C Din	0.1	K	12
Deltax	0	tempo	609.06	Px	1	Pmause	729

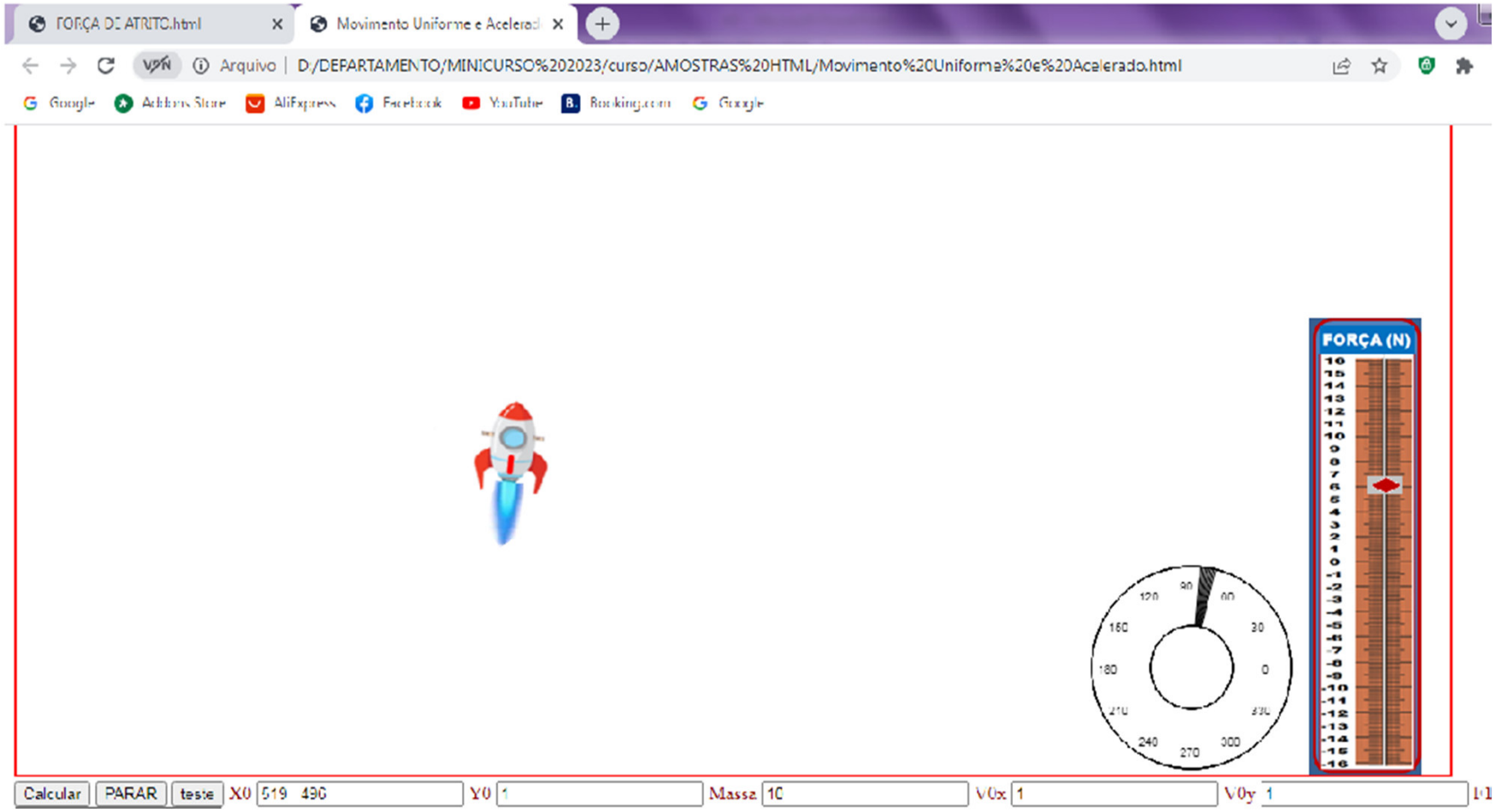
4.4 nterpolacao_Numerica



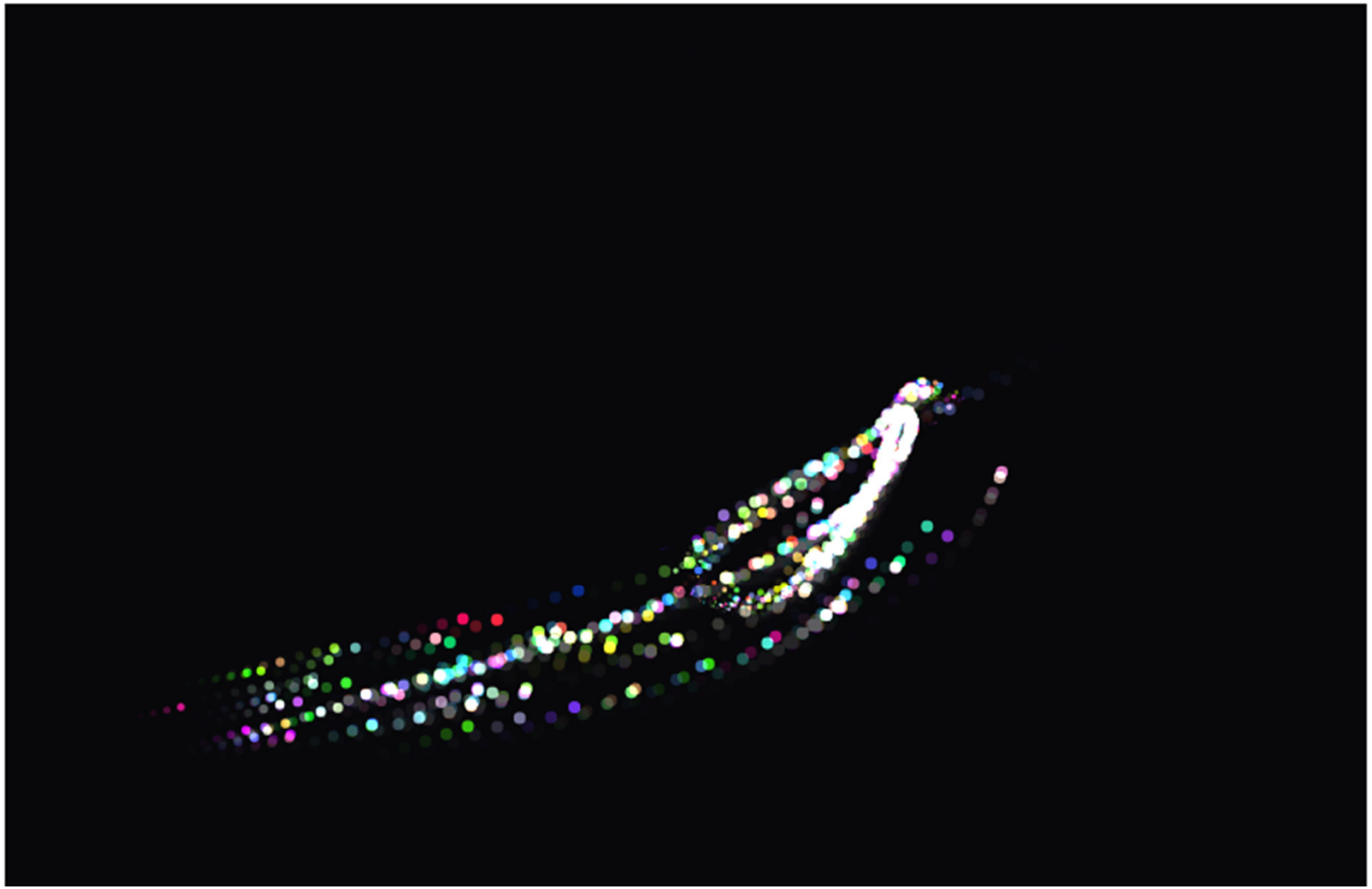
4.5 Movimento de Partículas



4.6 Movimento Uniforme e Acelerado

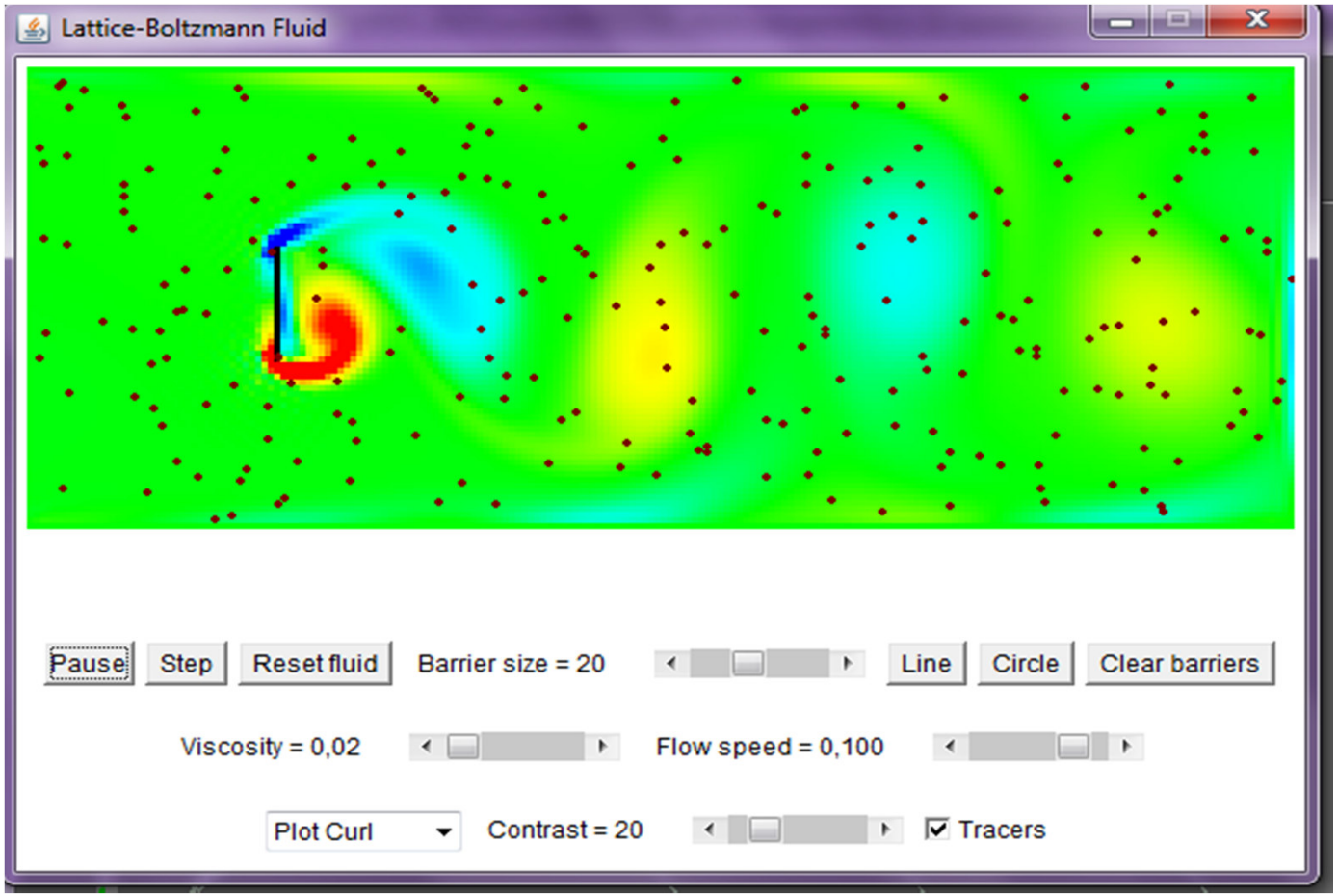


Movimento de Partículas 2



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.manualdocodigo.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