



01

EXPERIÊNCIA CRIATIVA NAVEGANDO NA COMPUTAÇÃO

PROJETO 2 — APLICATIVO MULTIMÍDIA

Luiz Antonio Pavão
Ciência da Computação
Escola Politécnica

01

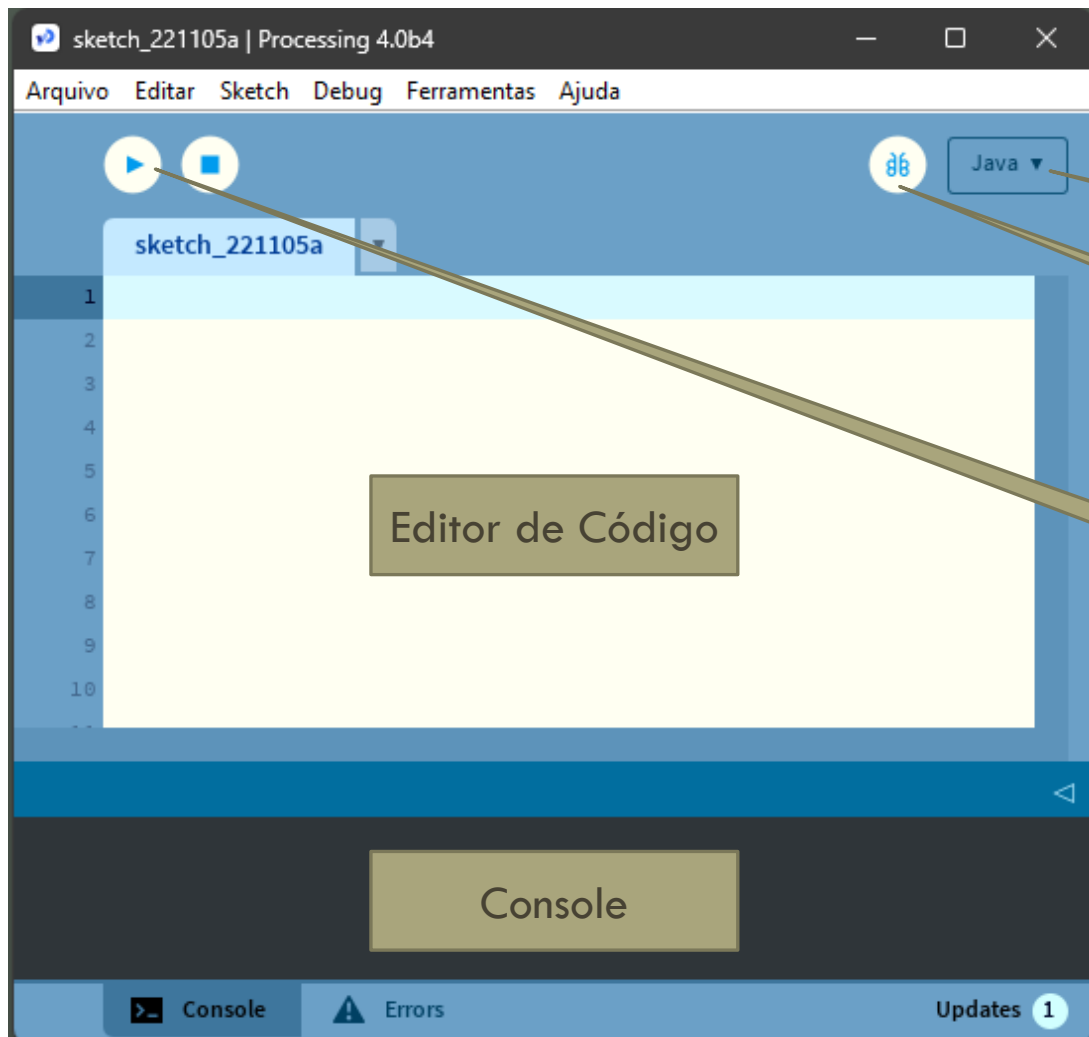
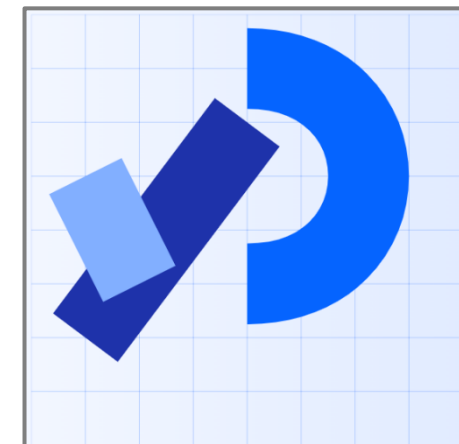
AMBIENTE DE DESENVOLVIMENTO & LINGUAGEM DE PROGRAMAÇÃO

IDE PROCESSING — RECURSOS, MODOS E FUNÇÕES

IDE Processing
Static & Active Modes
Variáveis, Tipos de Dados e
Funções

PROCESSING

ambiente integrado de desenvolvimento – IDE



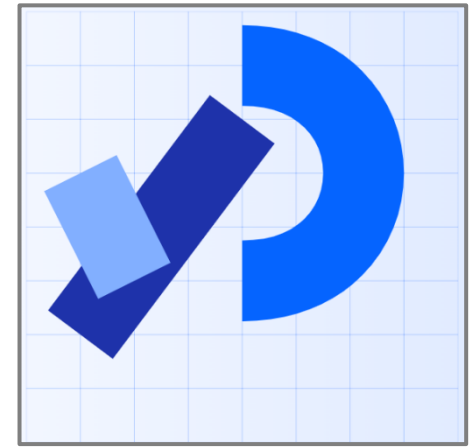
Editor de Código

Depuração
Debug

Executar

Linguagem
corrente

PROCESSING



<https://processing.org/>

“Processing is a flexible software sketchbook and a language for learning how to code. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology. There are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning and prototyping.”

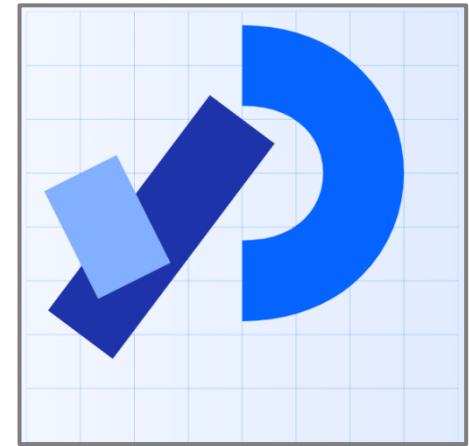
Processing é um sketchbook de software flexível e uma linguagem para aprender a codificar. Desde 2001, a Processing promove a alfabetização de software nas artes visuais e a alfabetização visual na tecnologia. Existem dezenas de milhares de estudantes, artistas, designers, pesquisadores e hobistas que usam Processing para aprendizado e prototipagem.

Processing é um projeto aberto iniciado por **Ben Fry** e **Casey Reas**.

É desenvolvido por uma equipe de voluntários em todo o mundo.

O ambiente de desenvolvimento (IDE) do **Processing** é “open source under the GPL”.

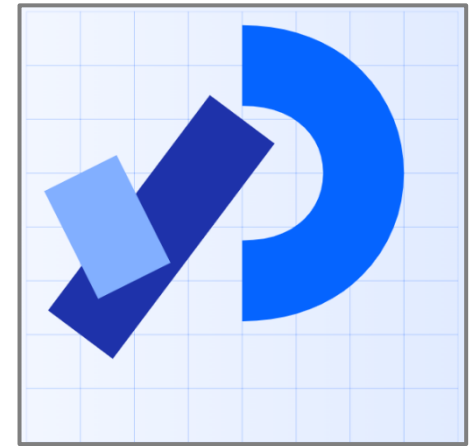
PROCESSING



O **Processing** também depende (e está em dívida com) outros projetos abertos, a saber:

- o **jEdit Syntax**, que é de domínio público;
- o compilador **ECJ** do projeto Eclipse, que usa a licença Eclipse;
- o projeto **Java Native Access** (JNA), lançado sob a LGPL;
- a partir da versão 0149, é usada uma versão ligeiramente modificada do **launch4j** para criar o processing.exe no Windows;
- a biblioteca **quaqua**, que torna os aplicativos Java mais parecidos com aplicativos Mac quando executados no OS X.

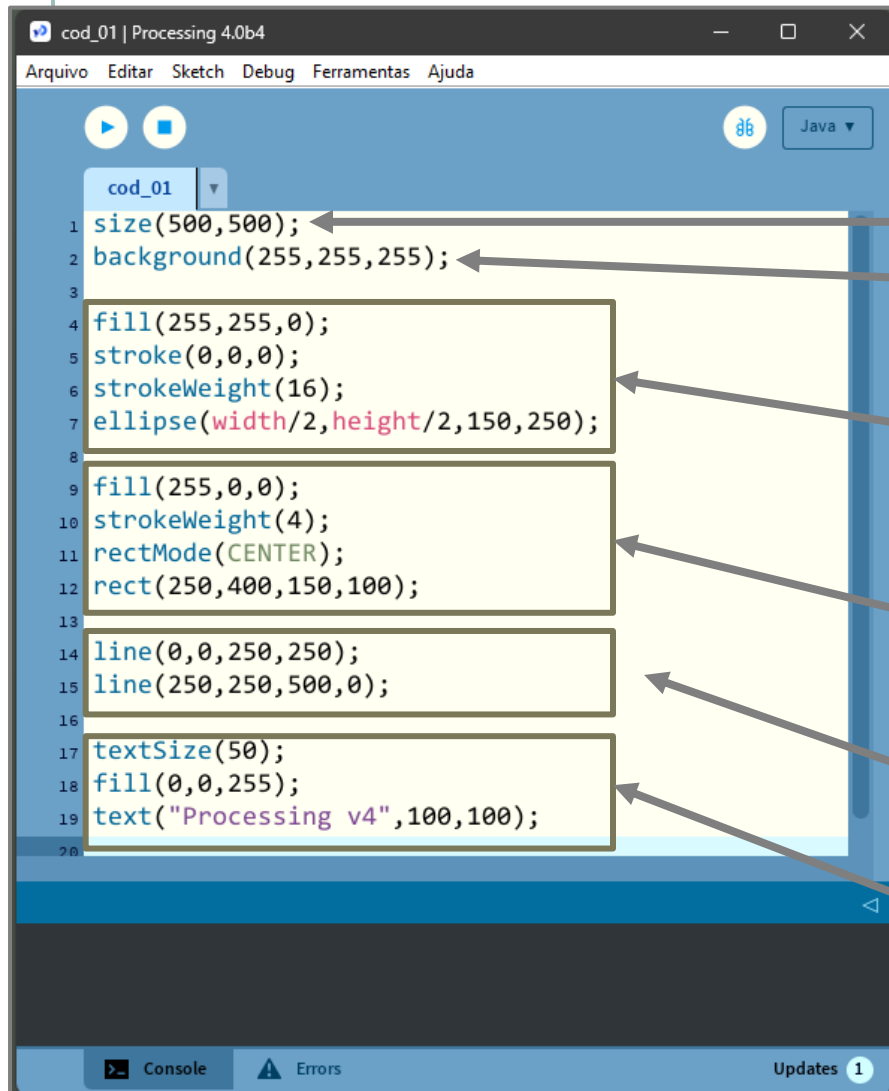
PROCESSING, JAVA



Java é um bom ponto de partida para uma *sketching language* porque é muito mais tolerante do que C++ e também permite que os usuários exportem suas aplicações para distribuição em muitas plataformas diferentes. “Quando começamos em 2001, a maioria das pessoas o usava para construir applets que rodavam na web, o que foi importante para o crescimento inicial do projeto.” [Ben Fry e Casey Reas]

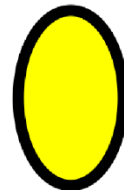
Processing não pretende ser o ambiente ou a linguagem final (na verdade, a linguagem é apenas Java, mas com uma nova API de gráficos e utilitários, além de algumas simplificações). “Fundamentalmente, o **Processing** apenas reúne nossa experiência na construção de coisas e tenta simplificar as partes que achamos que deveriam ser mais fáceis.” [Ben Fry e Casey Reas]

PROCESSING - INTERFACE, RECURSOS, COMANDOS, ETC...

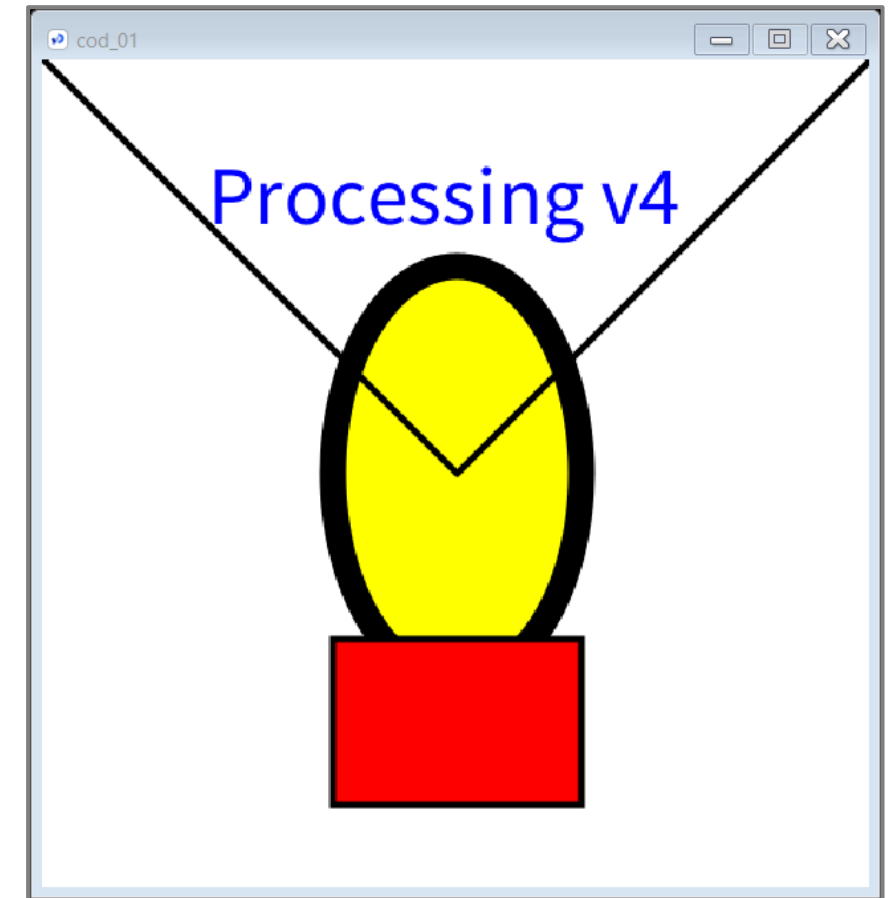


Tamanho do Canvas

Cor do fundo



Processing v4



SISTEMA DE UNIDADES E COORDENADAS

```
cod_01 | Processing 4.0b4
Arquivo  Editar  Sketch  Debug  Ferramentas  Ajuda

cod_01
1 size(500,500);
2 background(255,255,255);
3
4 fill(255,255,0);
5 stroke(0,0,0);
6 strokeWidth(16);
7 ellipse(width/2,height/2,150,250);
8
9 fill(255,0,0);
10 strokeWidth(4);
11 rectMode(CENTER);
12 rect(250,400,150,100);
13
14 line(0,0,250,250);
15 line(250,250,500,0);
16
17 textSize(50);
18 fill(0,0,255);
19 text("Processing v4",100,100);
20
```

Origem (0,0)

Eixo Y

Eixo X

Processing v4

Altura da tela - canvas (height)

Unidades = PIXEL

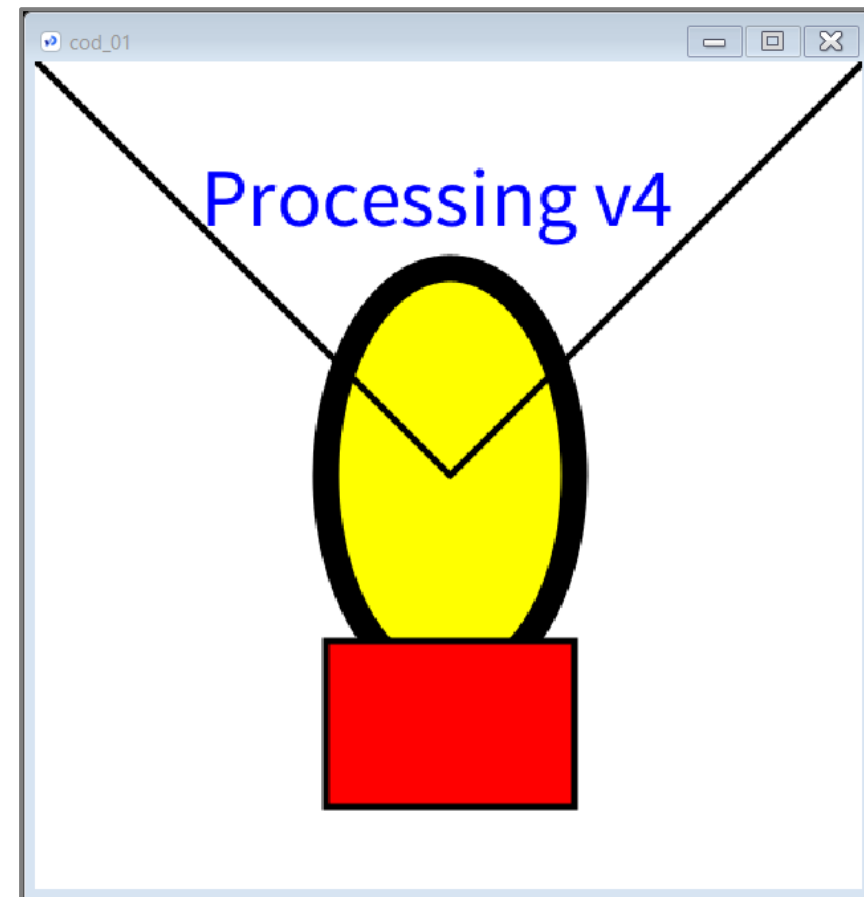
Largura da tela - canvas (width)

ORDEM DE DESENHO



```
1 size(500,500);
2 background(255,255,255);
3
4 fill(255,255,0);
5 stroke(0,0,0);
6 strokeWidth(16);
7 ellipse(width/2,height/2,150,250);
8
9 fill(255,0,0);
10 strokeWidth(4);
11 rectMode(CENTER);
12 rect(250,400,150,100);
13
14 line(0,0,250,250);
15 line(250,250,500,0);
16
17 textSize(50);
18 fill(0,0,255);
19 text("Processing v4",100,100);
```

**A ordem de
desenho segue a
ordem de
processamento**



PROCESSING - CORES / MODELO RGB

```
cod_01 | Processing 4.0b4
Arquivo Editar Sketch Debug Ferramentas Ajuda

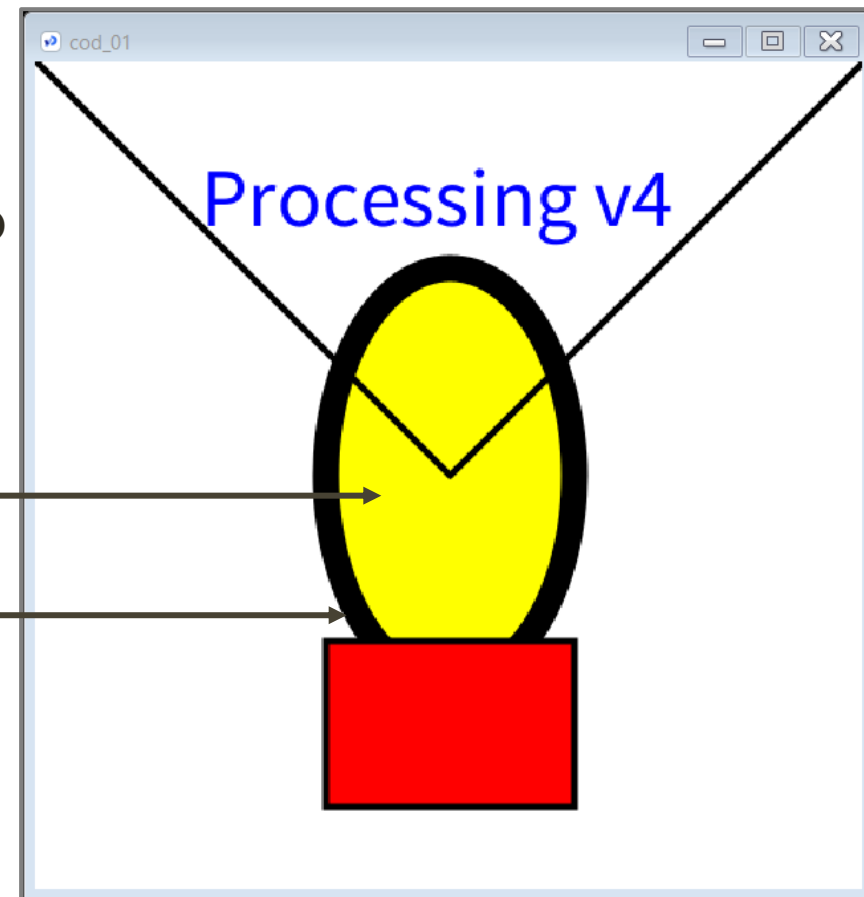
cod_01
1 size(500,500);
2 background(255,255,255);
3
4 fill(255,255,0);
5 stroke(0,0,0);
6 strokeWidth(16);
7 ellipse(width/2,height/2,150,250);
8
9 fill(255,0,0);
10 strokeWidth(4);
11 rectMode(CENTER);
12 rect(250,400,150,100);
13
14 line(0,0,250,250);
15 line(250,250,500,0);
16
17 textSize(50);
18 fill(0,0,255);
19 text("Processing v4",100,100);
20
```

PREENCHIMENTO

AMARELO
RED = 255
GREEN = 255
BLUE = 255

CONTORNO

PRETO
RED = 0
GREEN = 0
BLUE = 0



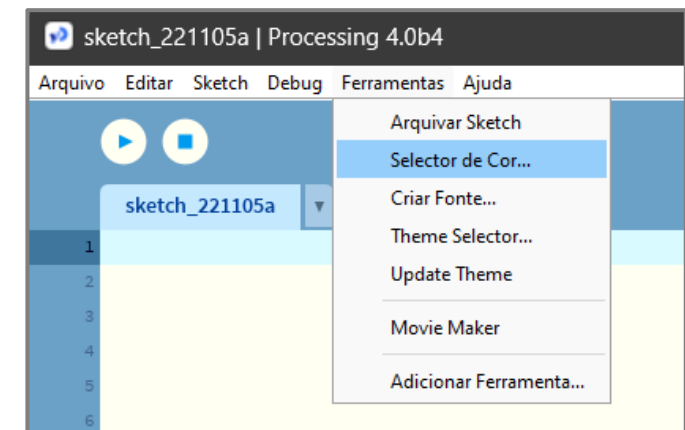
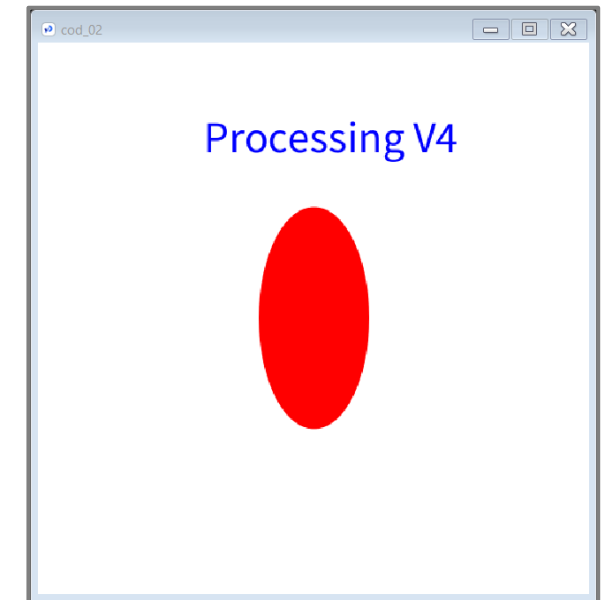
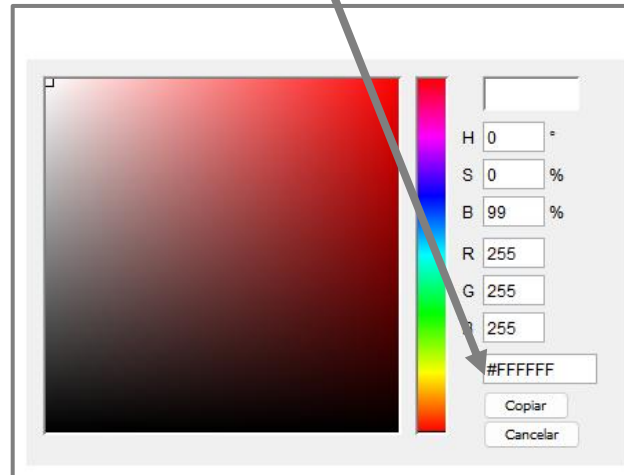
PROCESSING – VARIÁVEIS, SEMPRE TIPIFICADAS

```
cod_02 | Processing 4.0b4
Arquivo Editar Sketch Debug Ferramentas Ajuda

cod_02
1 int a = 100;
2 float b = 200.90;
3 double c = 1.1234567D;
4 boolean d = true;
5 color cor1 = color(255,0,0);
6 color cor2 = #0000FF;
7 char e = '4';
8 String texto = "Processing";
9
10 size(500,500);
11 background(255,255,255);
12 noStroke();
13
14 fill(cor1);
15 ellipse(250,250,a,b);
16
17 if (d) println(c); else println(d);
18
19 fill(cor2);
20 textSize(40);
21 text(texto + " V" + e, 150,100);
22
23
1.1234567
Console Errors Updates 1
```

Cor como parâmetro da função fill(), preenchimento

Cor em hexadecimal



PROCESSING – TIPOS DE DADOS

Definição

Os tipos de dados definem quais atribuições uma dada variável pode receber.

Java é uma linguagem tipificada, isto significa que ao ser definida devemos dizer a qual o tipo de dado ela é relacionada.

Em Java temos 8 tipos de dados primitivos:

byte, short, int, long, boolean, char, float e double

Em **Processing**, temos o tipo de dado **color** a ser adicionado a lista dos tipos primitivos.

PROCESSING – TIPOS DE DADOS

Alguns exemplos

Valores **booleanos** podem assumir somente **true** e **false**;

Valores do tipo **byte**, podem assumir números inteiros de **-128** até **127**;

Valores do tipo **char** recebem caracteres tipográficos (p.ex.: A, b, 1, \$);

Valores do tipo **int** recebem números inteiros de **-2.147.483.648** até **2.147.483.647**;

Valores do tipo **float** recebem números reais de **-1,4024E-37** até **3,4028234E+38**.

PROCESSING – TIPOS DE DADOS

Para o “computador” é importante a definição do tipo de dado, pois define a quantidade de memória necessária para armazenar as informações.

Na programação, devemos definir corretamente o tipo de dado. Isso é feito, levando em consideração o problema e seu contexto.

Exemplos:

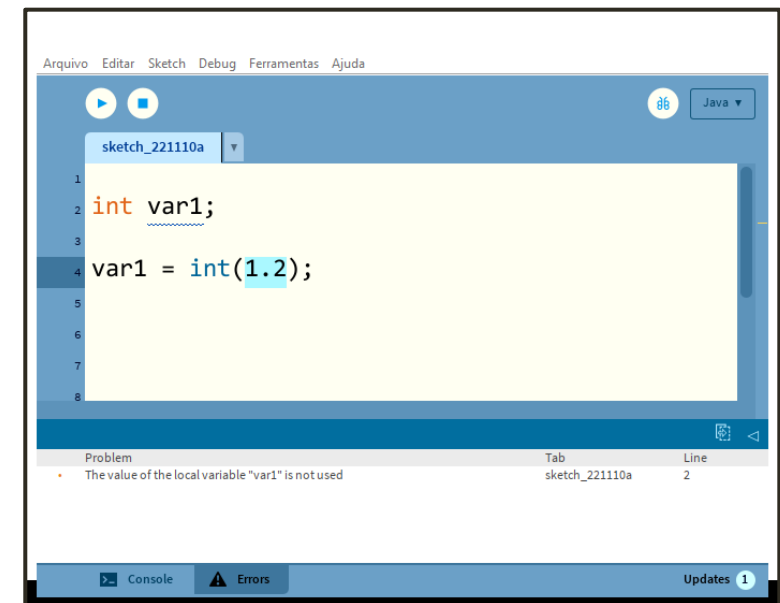
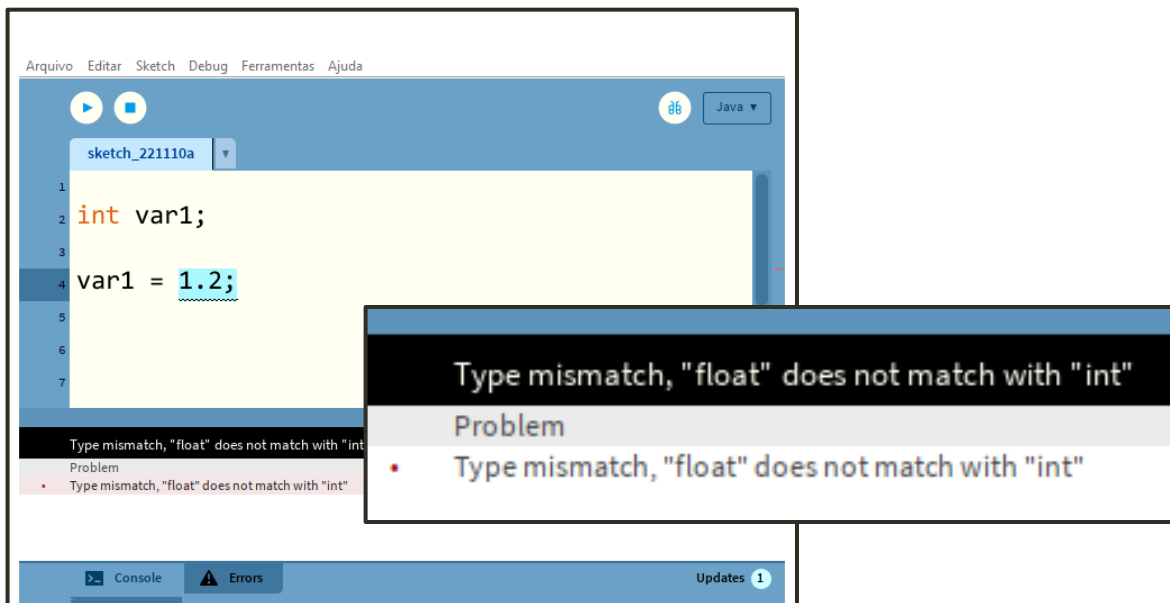
Para variáveis que armazenam coordenadas da tela, na maioria das vezes faz mais sentido usar inteiros. As unidades são pixel (picture elemento) que não são fracionários, não existe um meio pixel, eles variam de um em um.

Já para uma variável que represente a média de nossos gastos em um mês, deve ser float, caso queiramos considerar os centavos.

PROCESSING – TIPOS DE DADOS

Quando definimos uma variável com um certo tipo de dado, devemos sempre associar à ela uma atribuição deste mesmo tipo de dado. Se não tivermos este cuidado, erros podem ocorrer.

Existe um recurso chamado **casting** que permite a conversão de um tipo de dado em outro, mas deve fazer sentido para não criarmos mais problemas.



PROCESSING – COLOR DATA

Já vimos que as cores são um tipo de dado específico do Processing.

Podemos usar os modelos **RGB** (default) ou **HSB** (Hue-Saturation-Brightness / matiz-saturação-brilho).

As funções que recebem cor como parâmetro - **fill()**, **stroke()** e **background()** podem assumir diversas formas para recebe-los (modo **RGB** na função fill):

fill(rgb);

fill(rgb, alpha);

fill(gray);

fill(gray, alpha);

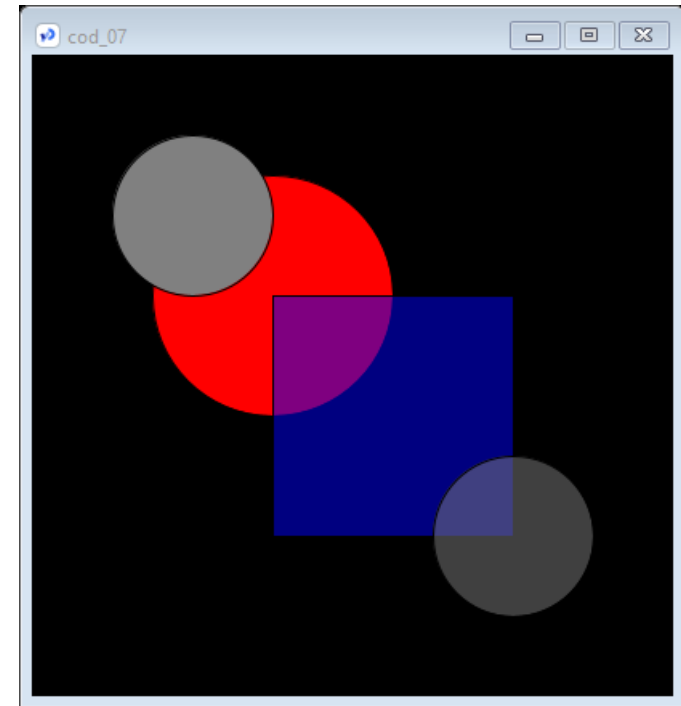
fill(v1, v2, v3);

fill(v1, v2, v3, alpha).

rgb	int: variável tipo <i>color</i> ou valor hexadecimal (web colors)
alpha	float: opacidade do preenchimento (transparência) (0 a 255)
gray	float: valor específico entre branco e preto (0 a 255)
v1	float: valor do componente Vermelho (0 a 255)
v2	float: valor do componente Verde (0 a 255)
v3	float: valor do componente Azul (0 a 255)

PROCESSING – COLOR DATA

```
cod_07
1 size(400,400);
2 background(0); // preto
3
4 fill(255,0,0); // vermelho
5 ellipse(150,150,150,150);
6
7 fill(0,0,255,128); // azul transparente
8 rect(150,150,150,150);
9
10 fill(128); // cinza
11 ellipse(100,100,100,100);
12
13 fill(128, 128); // cinza transparente
14 ellipse(300,300,100,100);
```



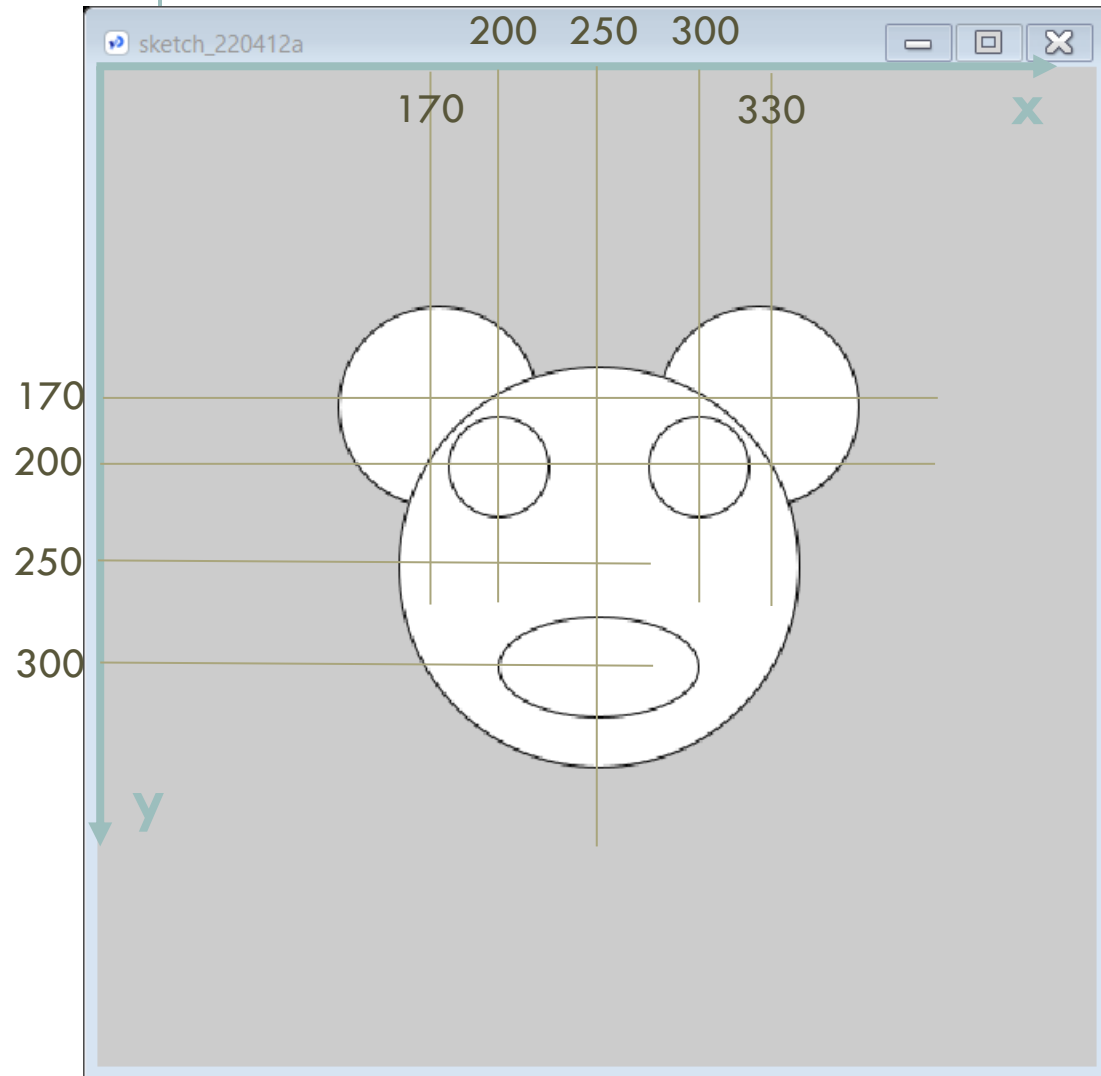
PROCESSING - OPERAÇÕES MATEMÁTICAS

```
cod_03 | Processing 4.0b4
Arquivo  Editar  Sketch  Debug  Ferramentas  Ajuda

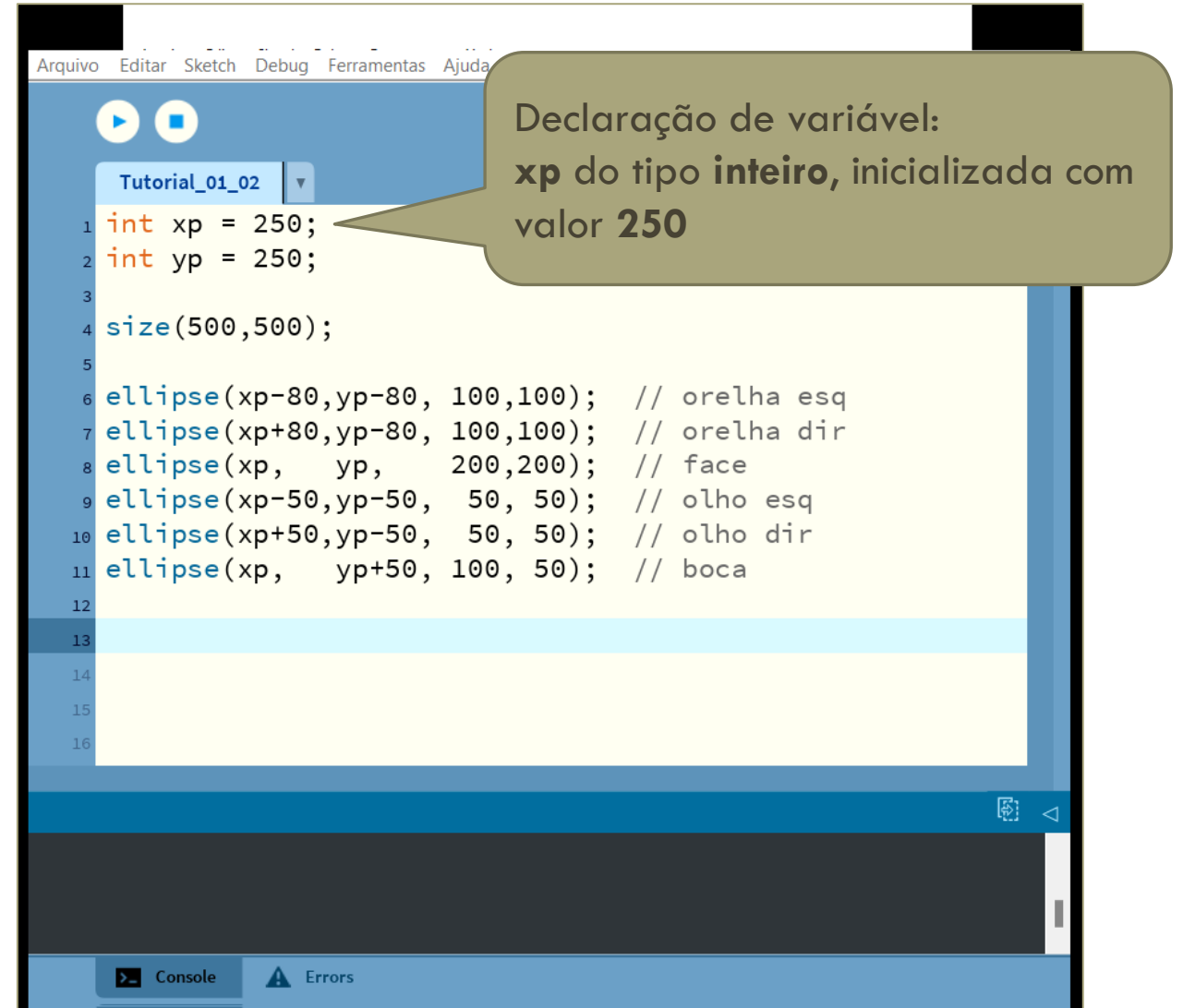
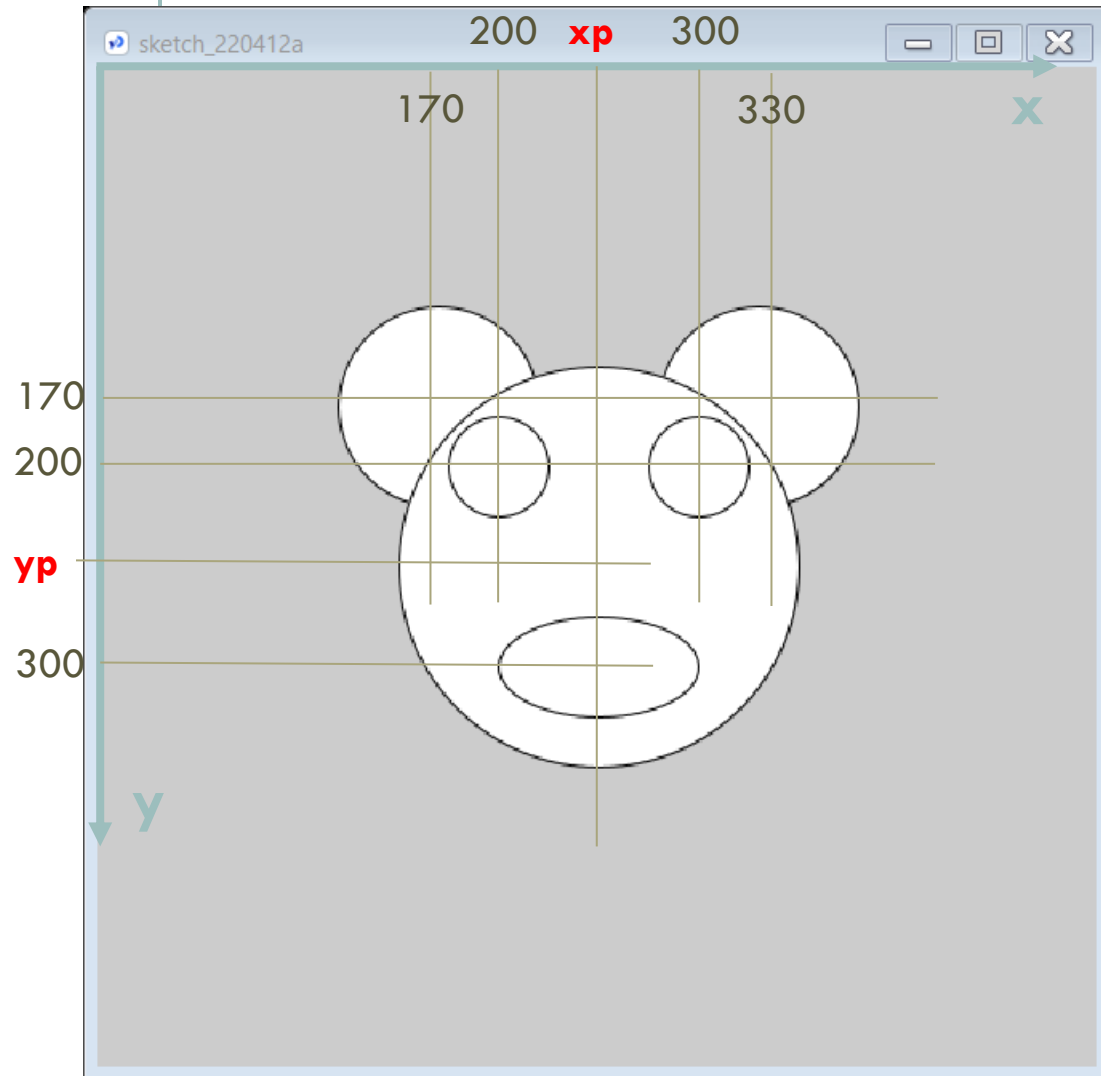
cod_03
1 int a = 600;
2 float b = 300.01;
3 double c = 1.1234567890123456789D;
4
5 int d;
6 float e, f;
7
8 size(500,500);
9 background(255,255,255);
10 fill(0);
11 textSize(35);
12
13 text("a = " + a,50,50);
14 text("b = " + b,50,100);
15 text("c = " + c,50,150);
16
17 d = int(a/b);
18 text("d = a / b = " + d,50,200);
19
20 e = sqrt(a);
21 text("e = raiz de (a) = " + e,50,250);
22
23 f = sin(b);
24 text("o seno de b é " + f,50,300);
25
26 float g = e/f;
27 text(g,50,350);
```

```
cod_03
a = 600
b = 300.01
c = 1.1234567890123457
d = a / b = 1
e = raiz de (a) = 24.494898
o seno de b é -0.9999269
-24,497
```

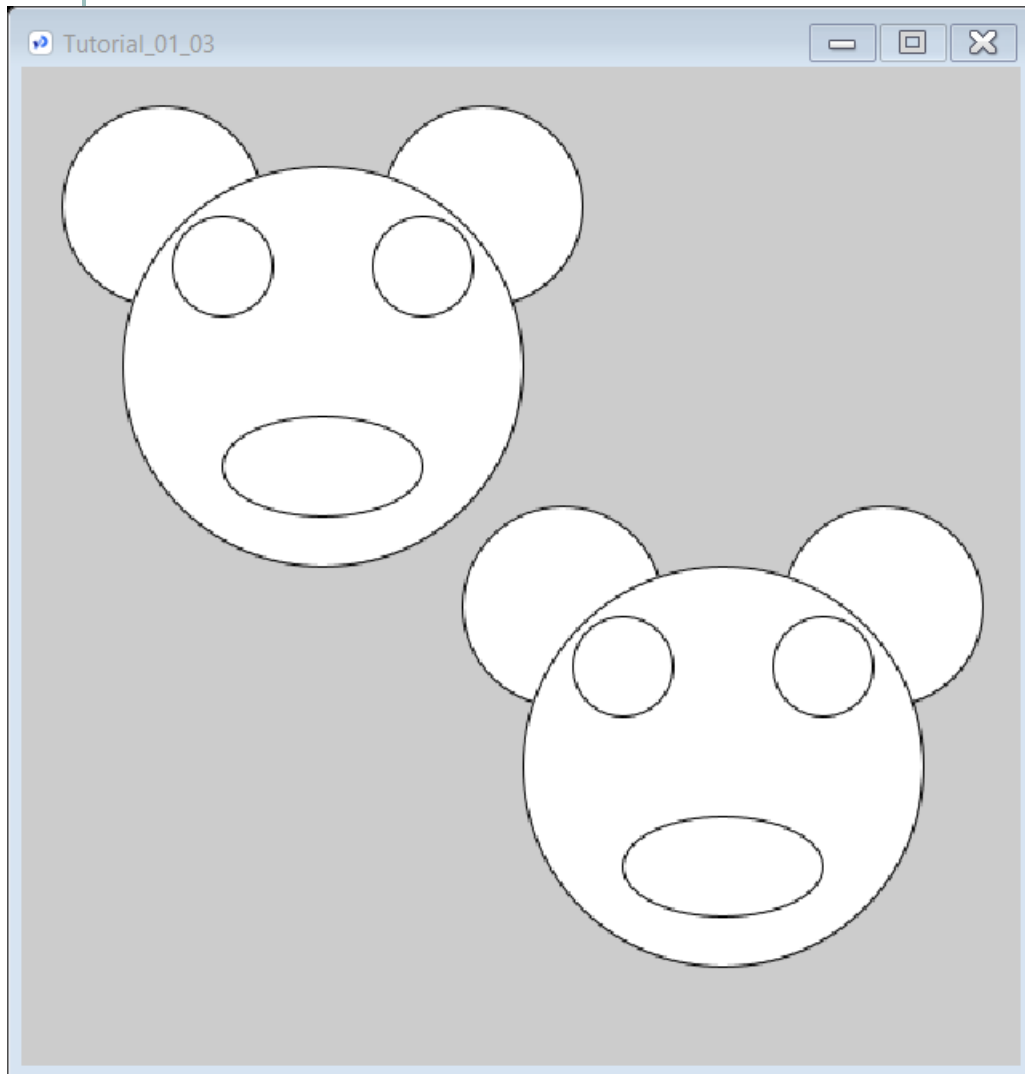
DESENHO EM COORDENADAS FIXAS



DESENHO EM COORDENADAS VARIÁVEIS



UMA SEGUNDA INSTÂNCIA DO DESENHO



Arquivo Editar Sketch Debug Ferramentas Ajuda

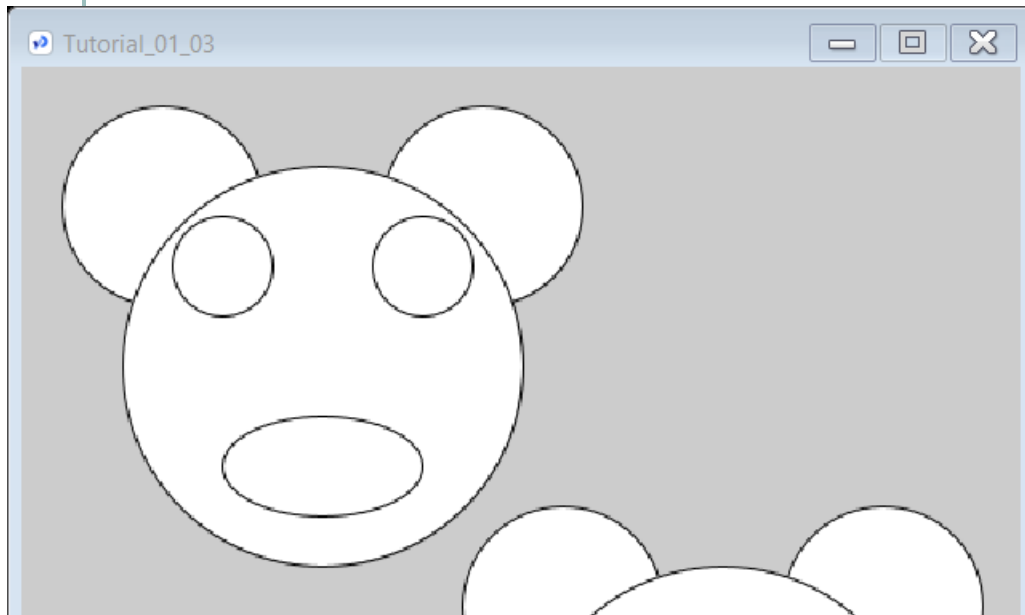
Tutorial_01_03

```
1 int xp = 150;
2 int yp = 150;
3
4 size(500,500);
5
6 ellipse(xp-80,yp-80, 100,100); // orelha esq
7 ellipse(xp+80,yp-80, 100,100); // orelha dir
8 ellipse(xp, yp, 200,200); // face
9 ellipse(xp-50,yp-50, 50, 50); // olho esq
10 ellipse(xp+50,yp-50, 50, 50); // olho dir
11 ellipse(xp, yp+50, 100, 50); // boca
12
13 xp = 350;
14 yp = 350;
15
16 ellipse(xp-80,yp-80, 100,100); // orelha esq
17 ellipse(xp+80,yp-80, 100,100); // orelha dir
18 ellipse(xp, yp, 200,200); // face
19 ellipse(xp-50,yp-50, 50, 50); // olho esq
20 ellipse(xp+50,yp-50, 50, 50); // olho dir
21 ellipse(xp, yp+50, 100, 50); // boca
22
23
```

Console Errors

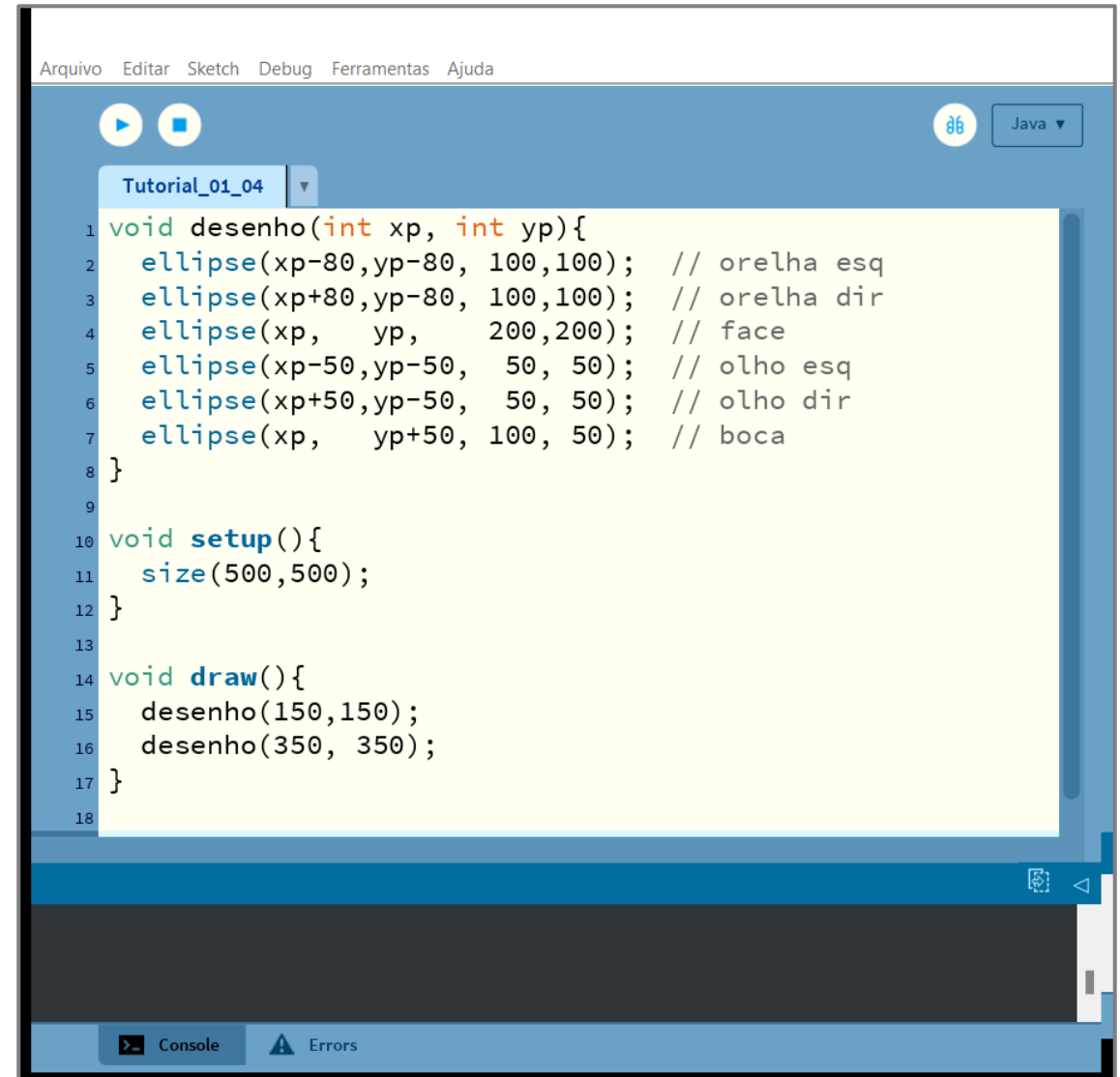
Redefinição dos valores das variáveis responsáveis pelo posicionamento do desenho

DESENHO COMO UMA FUNÇÃO – MODO ACTIVE



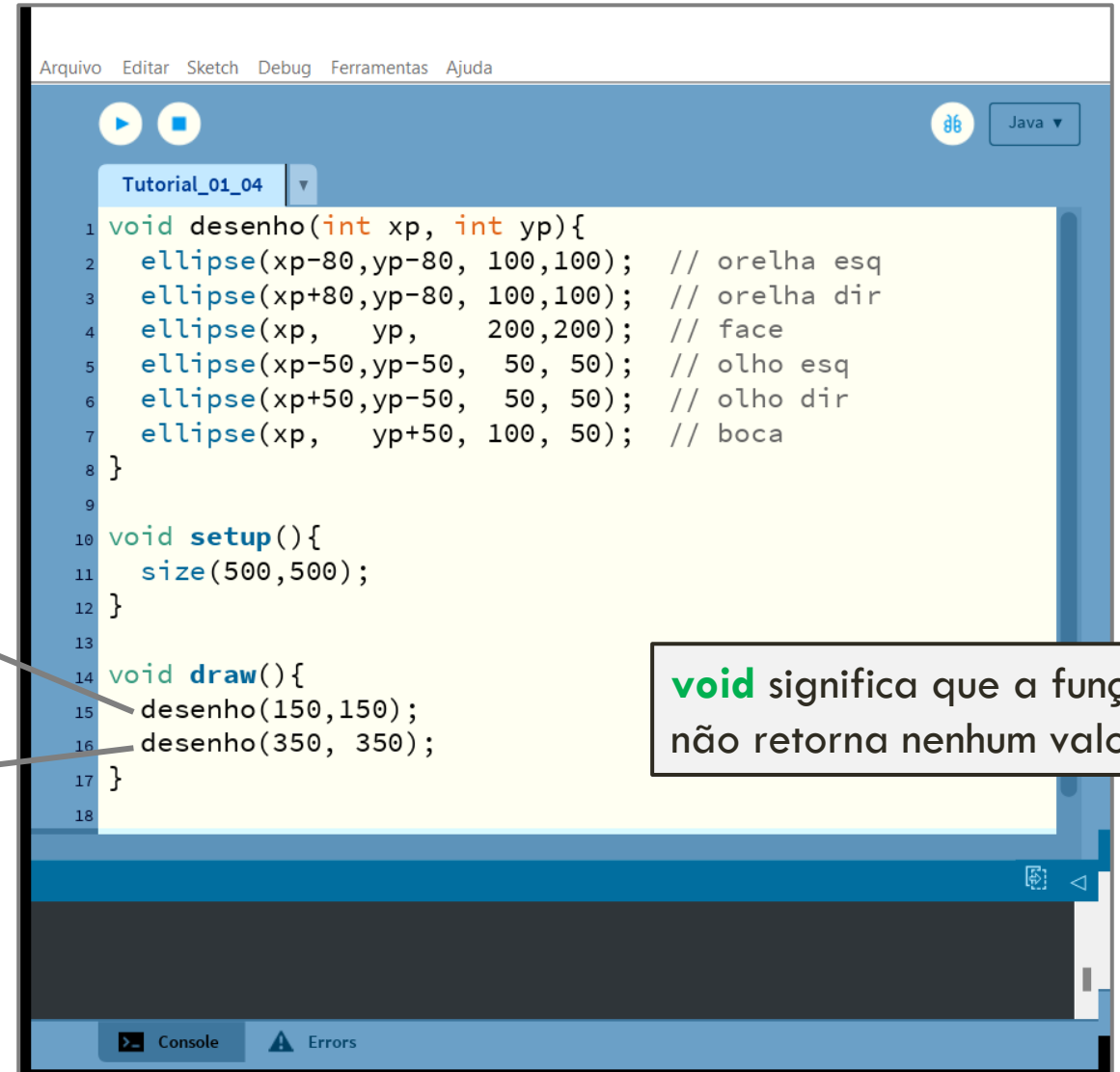
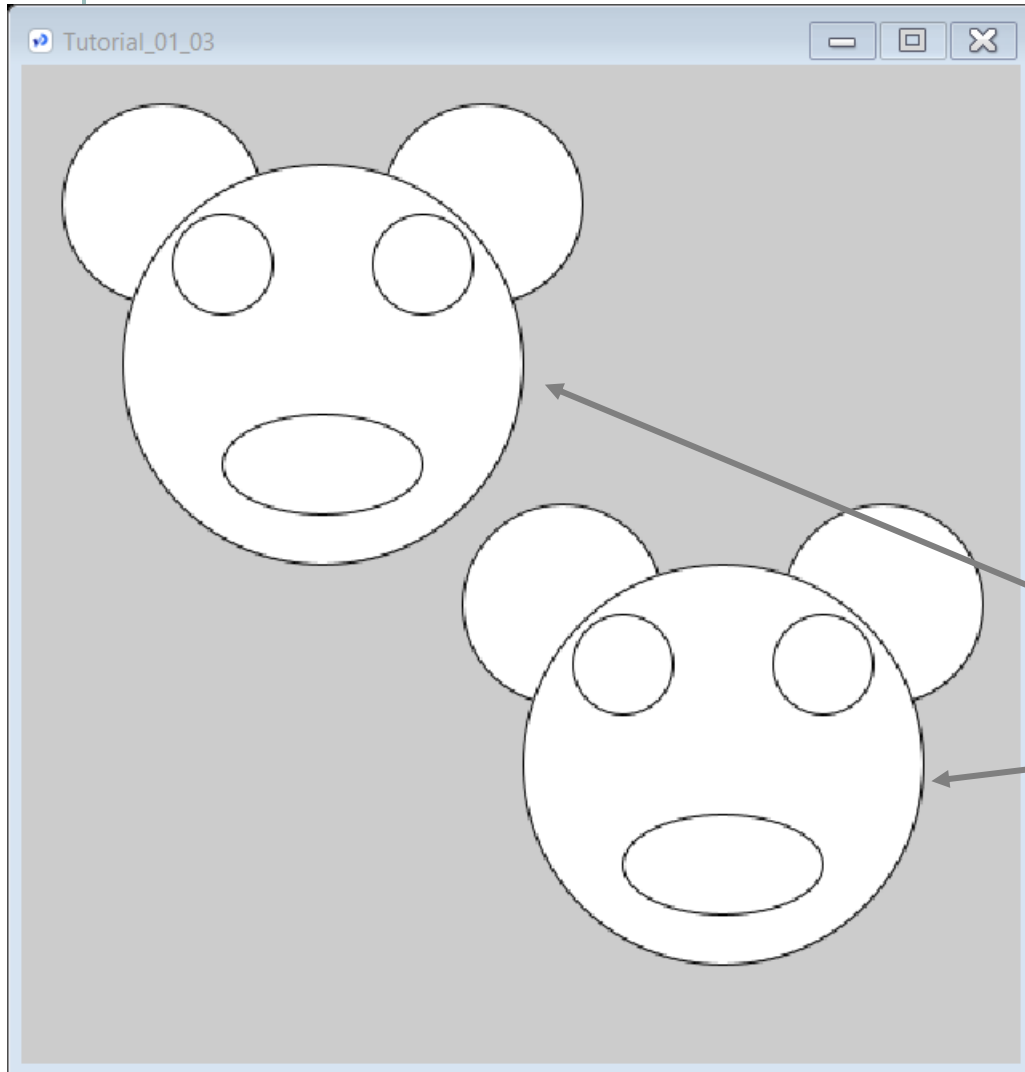
No modo **ACTIVE**, podemos criar nossas próprias funções. Para isso, devemos mudar a estrutura do código: **todo código deve pertencer a uma função**, com a exceção da declaração de variáveis globais.

Temos também as funções pré-definidas **setup()** e **draw()**. A princípio, colocamos tudo o que deve ser feito na inicialização dentro da função **setup()**, e tudo o que se refere a desenho, na função **draw()**.



```
1 void desenho(int xp, int yp){
2   ellipse(xp-80,yp-80, 100,100); // orelha esq
3   ellipse(xp+80,yp-80, 100,100); // orelha dir
4   ellipse(xp, yp, 200,200); // face
5   ellipse(xp-50,yp-50, 50, 50); // olho esq
6   ellipse(xp+50,yp-50, 50, 50); // olho dir
7   ellipse(xp, yp+50, 100, 50); // boca
8 }
9
10 void setup(){
11   size(500,500);
12 }
13
14 void draw(){
15   desenho(150,150);
16   desenho(350, 350);
17 }
18
```

DESENHO COMO UMA FUNÇÃO – MODO ACTIVE



FUNÇÃO COM RETORNO – MODO ACTIVE

```
cod_04 | Processing 4.0b4
Arquivo Editar Sketch Debug Ferramentas Ajuda

cod_04
1 void setup(){
2   size(500,500);
3   background(255,255,255);
4   textSize(30);
5   fill(0,0,0);
6 }
7
8 float rad_grau(float angulo){
9   return angulo*180/PI;
10 }
11
12 float grau_rad(float angulo){
13   return angulo*PI/180;
14 }
15
16 void draw(){
17   text("1 Graus é " + grau_rad(1) + " Radianos", 25,50);
18   text("1 Radianos é " + rad_grau(1) + " Graus", 25,100);
19   text("45 graus é " + grau_rad(45) + " Radianos", 25,150);
20   text("O seno de 45 graus é " + sin(grau_rad(45)), 25, 200);
21 }
```

1 Graus é 0.017453292 Radianos

1 Radianos é 57.295776 Graus

45 graus é 0.7853982 Radianos

O seno de 45 graus é 0.70710677

Funções trigonométricas
sempre recebem
argumentos em
RADIANOS

MODOS: STATIC X ACTIVE

No modo **Static** não temos funções, as declarações de variáveis e as linhas de comando são escritas em um único bloco e processadas uma única vez. É uma estrutura de programação muito limitada, não permite animação e nem interação com o usuário. É usada para testar porções de código antes de serem incorporadas em um código mais complexo no modo Active.

O modo **Active**, como já citado, o código é organizado e estruturado através de funções. Fora das funções, podemos ter as variáveis globais (aquelas que podem ser usadas em qualquer parte do código e os comandos de inclusão de bibliotecas. As variáveis declaradas dentro das funções, são válidas somente dentro das mesmas, são chamadas de variáveis locais.

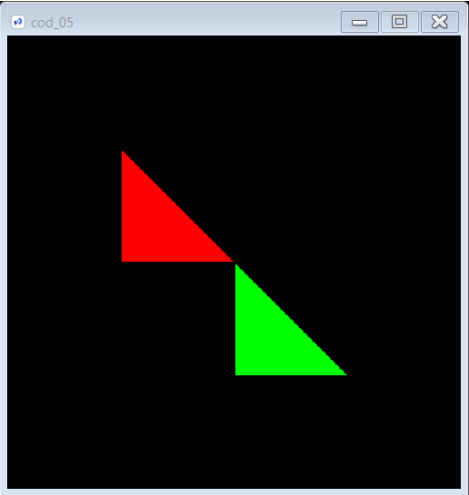
MODO ACTIVE

Variáveis Globais

Função para desenho de um triângulo retângulo alinhado com os eixos X e Y.
Parâmetros (argumentos da função):
coordenadas do vértice 90°, base, altura

Variáveis válidas
somente dentro da
função

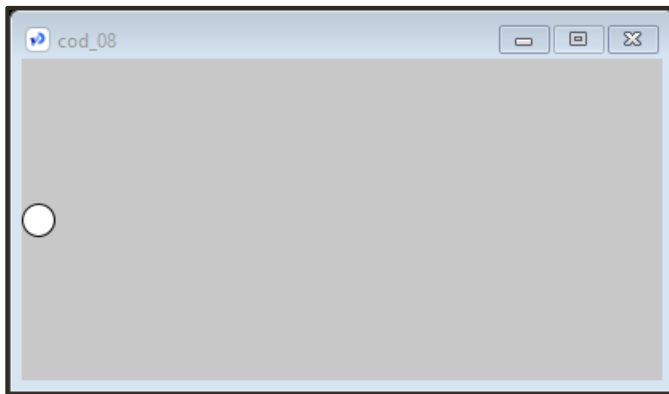
```
cod_05
1 int px = 100, py = 200;
2 int base = 100, altura = 100;
3
4 void triirect(int posX, int posY, int b, int a) {
5     triangle(posX, posY, posX+b, posY, posX, posY-a);
6 }
7
8 void setup(){
9     size(400,400);
10    background(0);
11 }
12
13 void draw(){
14     fill(255,0,0); // vermelho
15     triirect(px,py,base,altura);
16     fill(0,255,0); // verde
17     triirect(200,300,100,100);
18 }
```



MODOS: STATIC X ACTIVE

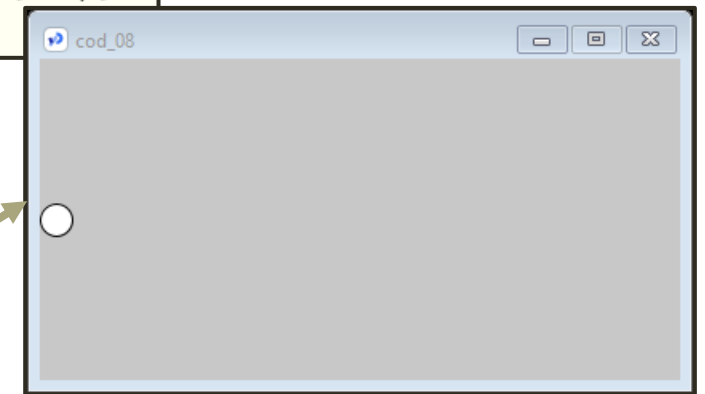
```
cod_08
1 int x = 10;
2 int y = 150;
3
4 size(400,200);
5
6 background(200);
7 ellipse(x,100,20,20);
```

Static



```
cod_08
1 int x = 10;
2 int y = 150;
3
4 void setup(){
5     size(400,200);
6 }
7
8 void draw(){
9     background(200);
10    ellipse(x,100,20,20);
11 }
```

Active

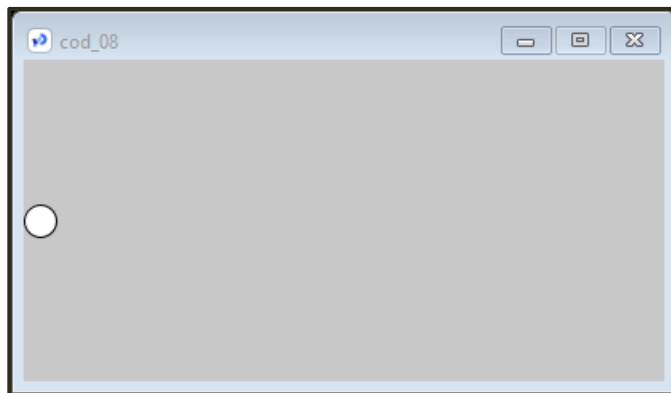


Resultados idênticos

MODULO STATIC

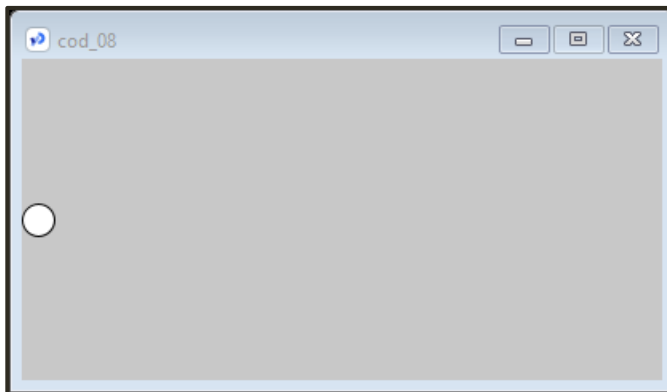
```
cod_08
1 int x = 10;
2 int y = 150;
3
4 size(400,200);
5
6 background(200);
7 ellipse(x,100,20,20);
```

O programa é executado apenas uma vez, os comandos são chamados na ordem do avanço das linhas.



MODO ACTIVE

```
cod_08
1 int x = 10;
2 int y = 150;
3
4 void setup(){
5     size(400,200);
6 }
7
8 void draw(){
9     background(200);
10    ellipse(x,100,20,20);
11 }
```



Duas funções devem estar presentes:

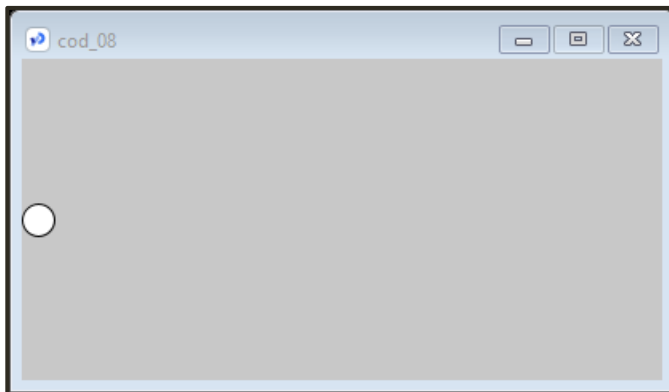
`setup()` e `draw()`

A função `setup()` é executada somente um vez e no início.

A função `draw()` é executada repetidamente, em um loop infinito.

MODO ACTIVE

```
cod_08
1 int x = 10;
2 int y = 150;
3
4 void setup(){
5     size(400,200);
6 }
7
8 void draw(){
9     background(200);
10    ellipse(x,100,20,20);
11 }
```

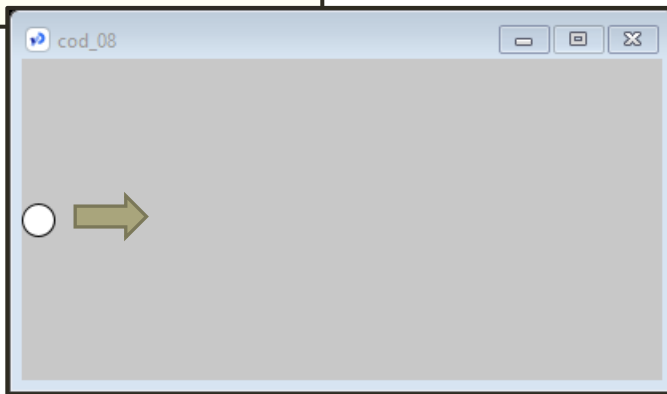


Na função `draw()`, a ellipse está sendo desenhada a partir de parâmetros fixos. Não há alteração nos valores das variáveis `x` e `y`.

Desta forma não há alteração, o resultado final é uma tela estática, mesmo estando no modo **Active**.

MODO ACTIVE - ANIMAÇÃO

```
cod_08
1 int x = 10;
2 int y = 150;
3
4 void setup(){
5     size(400,200);
6 }
7
8 void draw(){
9     background(200);
10    ellipse(x,100,20,20);
11    x = x + 1;
12 }
```



Para termos uma animação, (a elipse se movendo para a direita), é necessário que uma **variável** seja utilizada para a posição no eixo X.

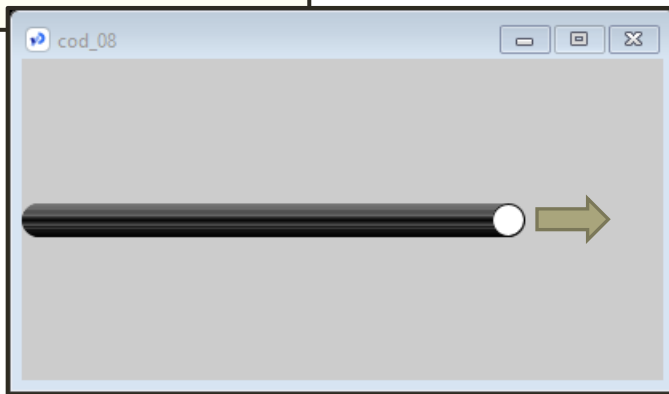
É preciso também que esta variável seja **incrementada** ($x = x + 1$) a cada loop do `draw()`.

Como a função `draw()` é chamada repetidamente (em loop), a variável `x` é incrementada continuamente, e consequentemente a posição da elipse é atualizada.

Repare que o comando `background` faz a limpeza da tela a cada chamada.

MODO ACTIVE - ANIMAÇÃO

```
cod_08
1 int x = 10;
2 int y = 150;
3
4 void setup(){
5   size(400,200);
6 }
7
8 void draw(){
9   // background(200);
10  ellipse(x,100,20,20);
11  x = x + 1;
12 }
```



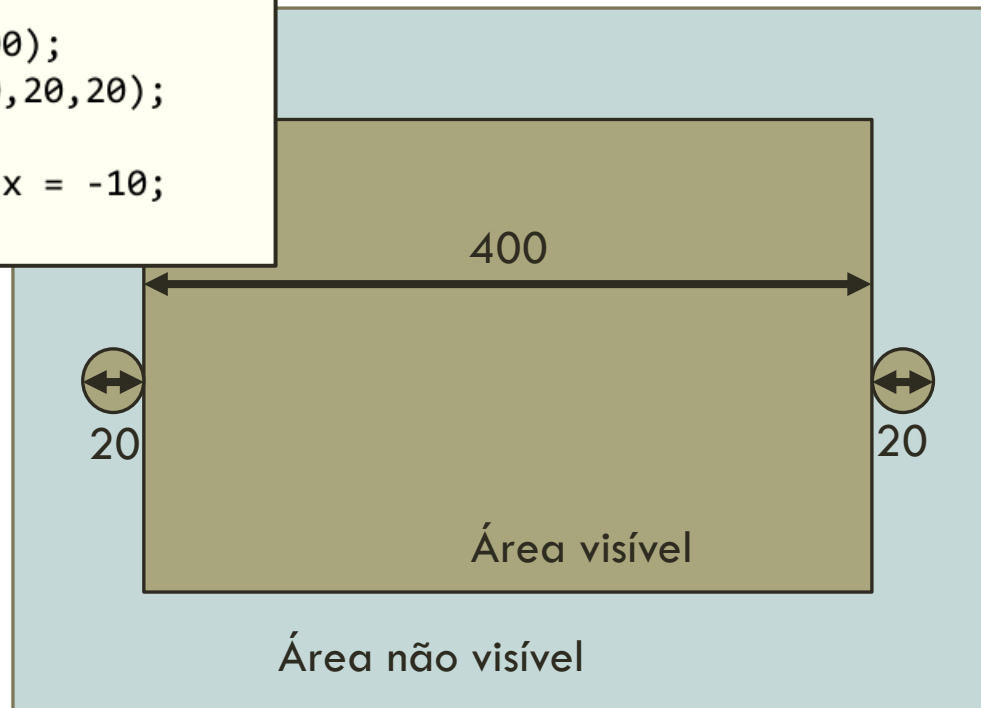
Neste caso, sem o comando `background()` sendo chamado, a atualização do desenho com as novas coordenadas, deixa um rastro.

Nos dois casos anteriores temos um problema: a elipse sai da tela ...

Como fazer para que ela saia, mas retorne ao ponto de partida e fique repetidamente fazendo este movimento?

MODO ACTIVE - ANIMAÇÃO

```
cod_08
1 int x = -10;
2 int y = 150;
3
4 void setup(){
5   size(400,200);
6 }
7
8 void draw(){
9   background(200);
10  ellipse(x,100,20,20);
11  x = x + 1;
12  if (x > 410) x = -10;
13 }
```



Simples:

Devemos controlar o valor da variável **x**, a que determina a posição da elipse.

Quando **x** ultrapassar o valor da largura da tela, deverá retornar a posição inicial:

```
if (x > 410) {x = -10;}
```

400 largura da tela

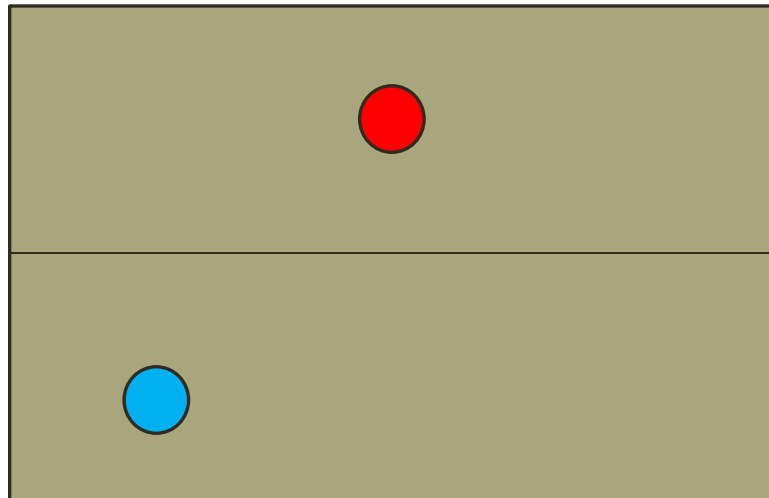
10 é o raio do círculo (começa e termina de ser desenhado quando estiver tangenciando a lateral da área visível - canvas).

COMANDO DE CONTROLE DE FLUXO — IF, ELSE

Muitas vezes desejamos que uma parte do código seja executada e outra não, dependendo de uma certa condição.

É exatamente isso que o comando if (se) faz.

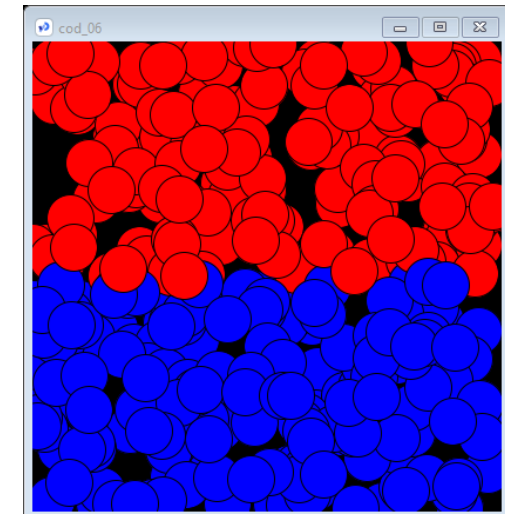
Para exemplificar seu uso, imagine uma situação em que dividimos a janela de desenho (canvas) horizontalmente em duas partes. Temos a geração de uma circunferência em coordenadas aleatórias.



COMANDO DE CONTROLE DE FLUXO – IF, ELSE

```
cod_06
1 float px, py;
2
3 void setup(){
4     size(400,400);
5     background(0);
6 }
7
8 void draw(){
9     px = random(0,width);
10    py = random(0,height);
11
12    if (py > height/2) fill(0,0,255); else fill(255,0,0);
13    ellipse(px,py, 40,40);
14 }
```

Se a variável py for maior do que a altura da tela dividida por dois (metade inferior do canvas) preencha com a cor azul, caso contrário preencha com a cor vermelha.



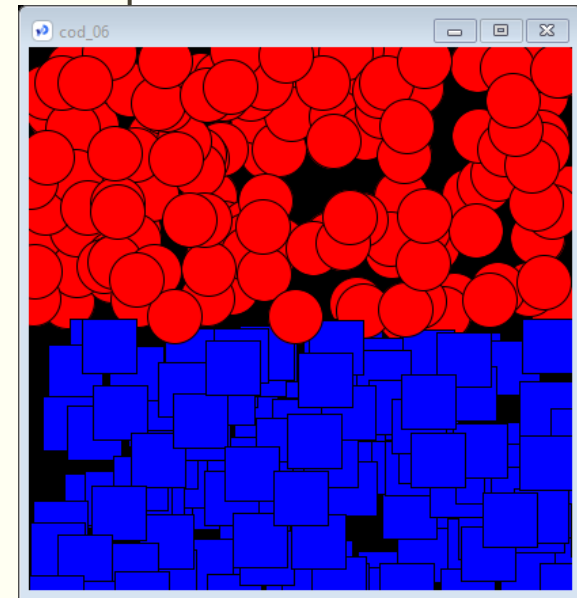
COMANDO DE CONTROLE DE FLUXO — IF, ELSE

Agora uma versão em que na porção inferior será desenhado um retângulo azul e na superior uma circunferência vermelha.

Como temos mais de um comando a ser utilizado, agora precisamos criar uma seção para o código.

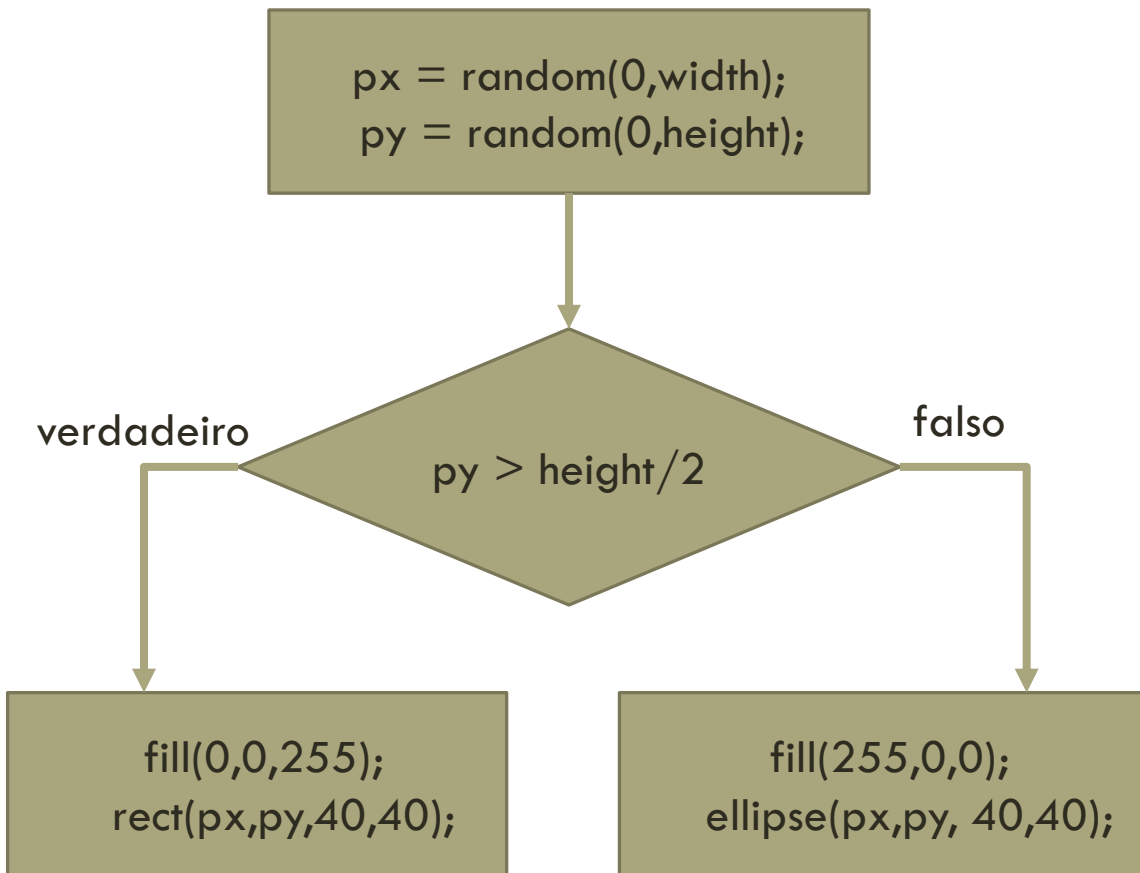
Isso é feito com os delimitadores { e }.

```
cod_06
1 float px, py;
2
3 void setup(){
4   size(400,400);
5   background(0);
6 }
7
8 void draw(){
9   px = random(0,width);
10  py = random(0,height);
11
12  if (py > height/2) {
13    fill(0,0,255);
14    rect(px,py,40,40);
15  }
16  else {
17    fill(255,0,0);
18    ellipse(px,py, 40,40);
19  }
20 }
```

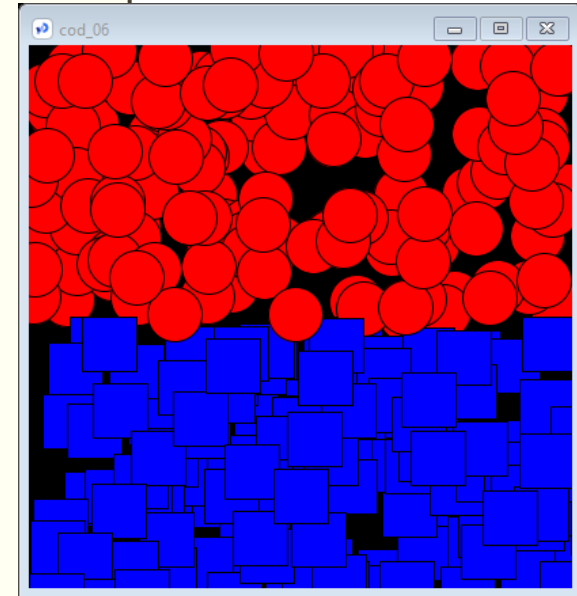


COMANDO DE CONTROLE DE FLUXO – IF, ELSE

Representações gráficas utilizando fluxogramas, são muito úteis para o processo de desenvolvimento do código (projeto).



```
cod_06
1 float px, py;
2
3 void setup(){
4   size(400,400);
5   background(0);
6 }
7
8 void draw(){
9   px = random(0,width);
10  py = random(0,height);
11
12  if (py > height/2) {
13    fill(0,0,255);
14    rect(px,py,40,40);
15  }
16  else {
17    fill(255,0,0);
18    ellipse(px,py, 40,40);
19  }
20 }
```



FUNÇÕES

- Pode ser definida como um conjunto de instruções que permitem processar comandos e variáveis para a obtenção de um determinado valor ou um comportamento do programa;
- Também pode ser denominada de Procedimento;
- Ela encapsula ações que tem por finalidade realizar uma atividade descrita por um algoritmo;
- Pode retornar um valor ou não;
- Pode receber parâmetros ou não;
- Em **Processing**, sempre no modo **ACTIVE**.

FUNÇÕES – EXEMPLOS, FUNÇÕES PROCESSING

Tipo de retorno
Neste caso: nenhum

Nome da Função

Parâmetros
Neste caso: 4

```
ellipse(100,100,200,250);
```

Tipo de retorno
Neste caso: float

Nome da Função

Parâmetros
Neste caso: um
ou dois

```
float r1 = random(50);  
float r2 = random(-50, 50);
```

float random(range)

Função que retorna um valor randômico do tipo **float**.

Nestes exemplos, entre 0 e 50 (r1) e -50 e +50 (r2), o valor retornado não inclui os limites do intervalo.

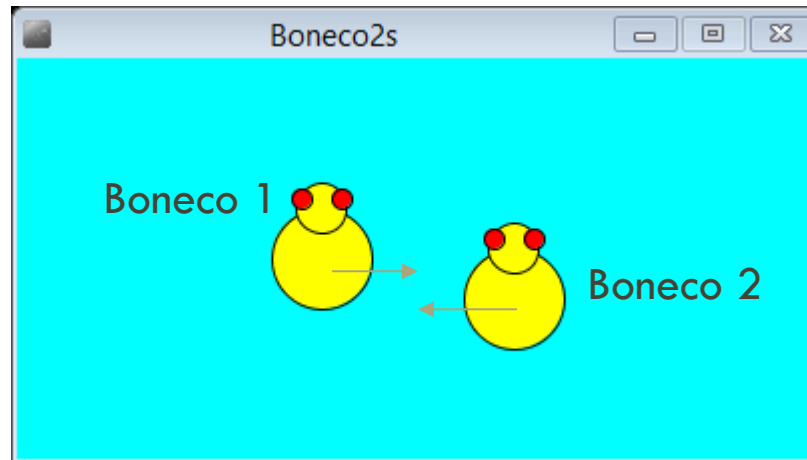
FUNÇÕES – EXEMPLOS, FUNÇÕES PROCESSING

Aqui um exemplo do uso das funções mencionadas, é criada uma ellipse com uma definição aleatória de sua cor de preenchimento:

```
fill(random(0,255), random(0,255),random(0,255));  
ellipse(200,200,150,150);
```


FUNÇÕES - MODO ACTIVE

Exemplo: Sem o uso de função



```
float Xp = 100;
float Xp2 = 50;
float Yp;

void setup() {
    size(400,200);
}

void draw() {
    background(0,255,255);

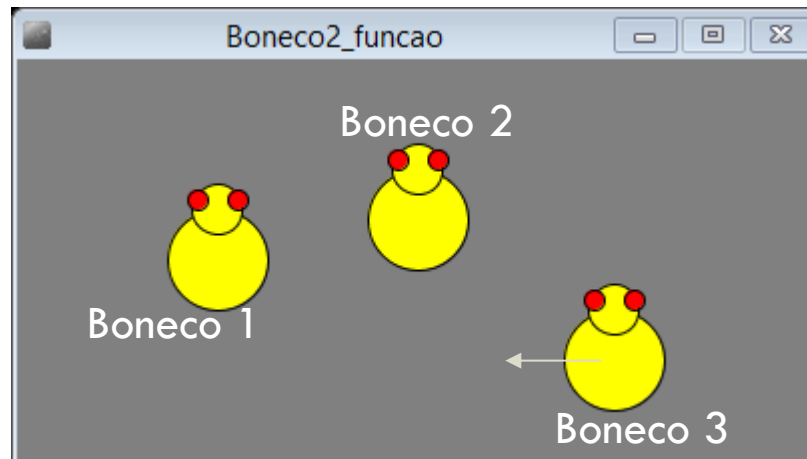
    // boneco1 (esquerda para direita)
    Yp = 100;
    fill(255,255,0);
    ellipse(Xp,Yp,50,50); // corpo
    ellipse(Xp,Yp-25,25,25); // cabeça
    fill(255,0,0);
    ellipse(Xp-10,Yp-30, 10,10); // olho esq
    ellipse(Xp+10,Yp-30, 10,10); // olho dir

    // boneco2 (direita para esquerda)
    Yp = 120;
    fill(255,255,0);
    ellipse(400-Xp,Yp,50,50);
    ellipse(400-Xp,Yp-25,25,25);
    fill(255,0,0);
    ellipse(400-Xp-10,Yp-30, 10,10);
    ellipse(400-Xp+10,Yp-30, 10,10);

    // incremento do valor da posição em X
    Xp = Xp + 1; // bonecos 1 e 2
    if (Xp>425) { Xp = -25; }
    Xp2 = Xp2 + 0.5; // boneco 3 mais lento
    if (Xp2>425) { Xp2 = -25; }
}
```

FUNÇÕES - MODO ACTIVE

Exemplo: Com o uso de função



```
float Xb = 0;
float Yb = 150;

void setup() {
    size(400,200);
}

void boneco(float Xp, float Yp) {
    fill(255,255,0);
    ellipse(Xp,Yp,50,50); // corpo
    ellipse(Xp,Yp-25,25,25); // cabeça
    fill(255,0,0);
    ellipse(Xp-10,Yp-30, 10,10); // olho esq
    ellipse(Xp+10,Yp-30, 10,10); // olho dir
}

void draw() {
    background(128);

    boneco(100,100); // boneco1
    boneco(200,80); // boneco2

    boneco(Xb,Yb); // boneco3
    Xb++;
    if (Xb>425) Xb = -25;
}
```

Parâmetros da função:
Posição (X,Y) do boneco