Programação I

George Gomes

Funções e Sub-rotinas

Até o momento, vimos comandos de:

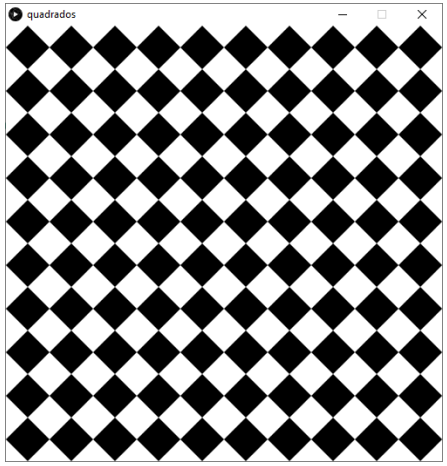
- declaração de variáveis
- atribuição de valores às variáveis
- desenho
- entrada e saída de dados
- desvios condicionais
- laços de repetição

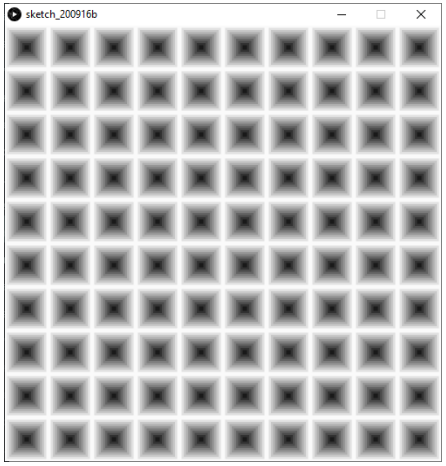
As estruturas já estudadas são suficientes para se escrever qualquer programa!

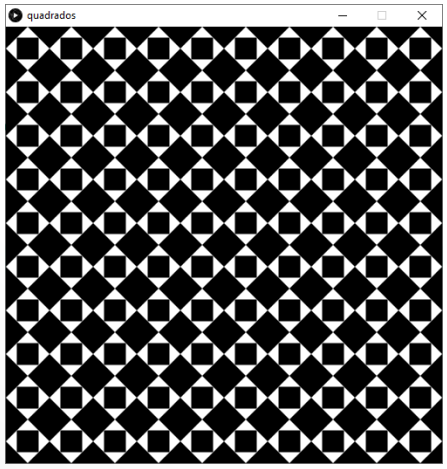
Mas, podemos fazer uso de mais um recurso para tornar a construção de algoritmos de maneira mais fácil, mais legível, mais organizada.

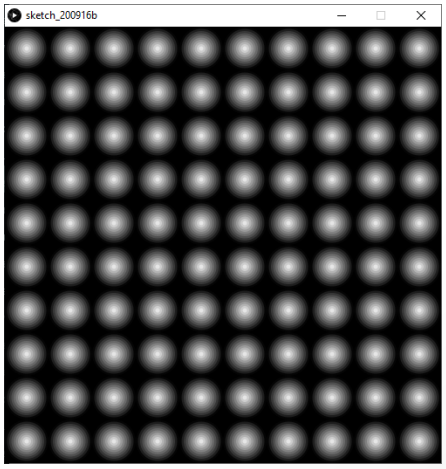
Para contextualizar, vamos considerar o exemplo a seguir:

O que essas peças tem em comum?









Todos esses exemplos usam o mesmo código abaixo, mas o que faz eles serem tão diferentes?

```
void setup() {
  size(500, 500);
  noStroke();
  background(255);
  rectMode(CENTER);
  for (int j=0; j<10; j++)
    for (int i=0; i<10; i++)
      desenho(i*50+25, j*50+25, 50);
}
```

A função de desenho!

```
void setup() {
  size(500, 500);
  noStroke();
  background(255);
  rectMode(CENTER);
  for (int j=0; j<10; j++)
    for (int i=0; i<10; i++)
      desenho(i*50+25, j*50+25, 50);
}
```

Os laços aninhados repetem 10 x 10 vezes a instrução de desenho para preencher toda a tela (10 linhas e 10 colunas)

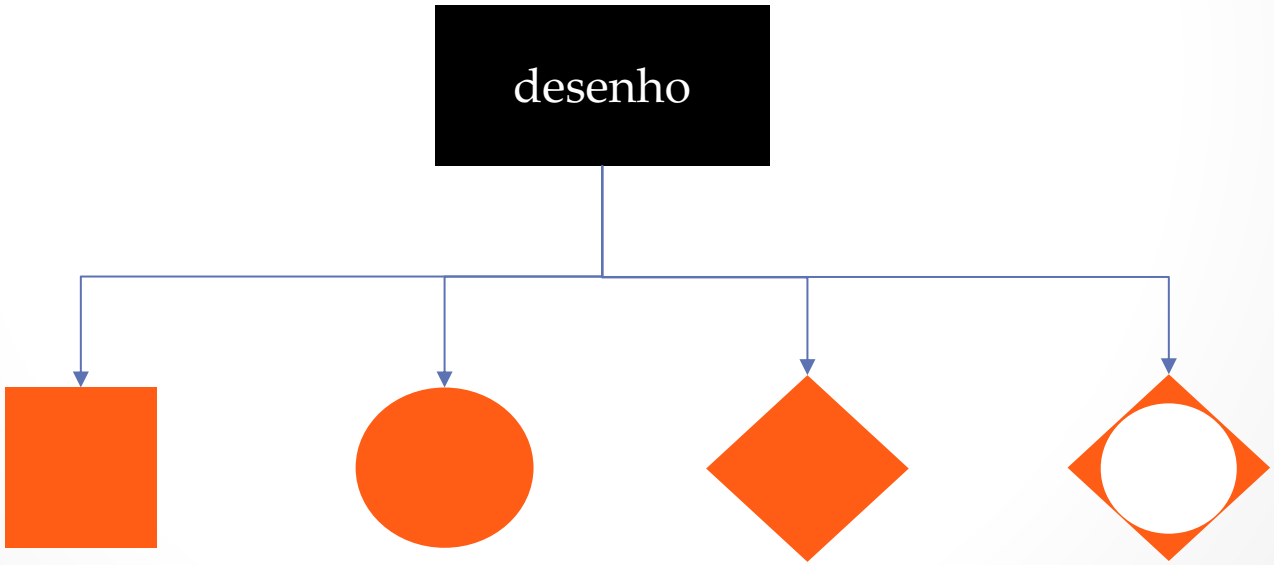
```
void setup() {
  size(500, 500);
  noStroke();
  background(255);
  rectMode(CENTER);
  for (int j=0; j<10; j++)
    for (int i=0; i<10; i++)
      desenho(i*50+25, j*50+25, 50);
}
```

Veja que com o mesmo código podemos desenhar vários padrões diferentes, apenas mudando a função de desenho

O código fica mais limpo, mais fácil de entender e reaproveitável

Mas como a função de desenho funciona?

Por enquanto ela funciona como uma caixa preta. Pode ser um *rect*, uma *ellipse*, um *quad*, ou uma combinação de vários comandos de desenho



Antes de escrever nossa função de desenho, vamos entender um pouco mais sobre funções

Para escrever o código da função, é necessário atentarmos para sua estrutura:

Pela figura do slide anterior, vemos que será necessário especificar:

- o tipo do valor de entrada
- o tipo do valor de saída

Para escrever nossas funções, seguiremos a sintaxe descrita abaixo:

```
<tipo_da_saída> nome_da_função (<lista_valores_entrada>){
      comando1;
      comando2;
      …
      comando de retorno de resultado;
}
```

A lista de valores de entrada será descrita por:

<tipo> <nome>

Se houver mais de um item na lista, eles são separados por vírgulas.

Exemplo, uma função que recebe como entrada dois inteiros e devolve o resultado da soma desses dois números

```
int soma(int n1, int n2) {
  int resultado;
  resultado = n1 + n2;
  return resultado;
}
```

Vamos analisar a sintaxe de uma função

Nome da função

```
int soma(int n1, int n2) {
   int resultado;
   resultado = n1 + n2;
   return resultado;
}
```

Vamos analisar o código

Tipo da saída

```
int soma(int n1, int n2) {
    int resultado;
    resultado = n1 + n2;
    return resultado;
}
```

Vamos analisar a sintaxe de uma função

Lista de valores e tipos da entrada

```
int soma(int n1, int n2) {
   int resultado;
   resultado = n1 + n2;
   return resultado;
}
```

Vamos analisar a sintaxe de uma função

```
int soma(int n1, int n2) {
   int resultado;
   resultado = n1 + n2;
   return resultado;
}
```

Comando de retorno do resultado

O comando ***return*** deve ser o último comando

```
int soma(int n1, int n2) {
   int resultado;
   resultado = n1 + n2;
   return resultado;
}
```

Uma vez escrita a função, vamos ver agora como usá-la! Costumamos falar que vamos chamar a função

chama a função soma(), implementada no exemplo

```
int num;
num = soma(3, 5);
```

Outro exemplo, uma função que verifica se um número é par

```
boolean ehPar(int n) {
  boolean resultado;
  if( n%2==0) resultado = true;
  else resultado = false;
  return resultado;
}
```

E uma função que desenha um círculo na tela? Qual seria o retorno desta função?

Veja essa palavra estranha *void*, a tradução é vazio, quer dizer que não retorna nada e não tem o comando ***return***

```
void circulo(int x, int y, int l) {
  ellipse(x, y, l, l);
}
```

O comando ***return*** só é necessário quando a função retorna valor

```
void circulo(int x, int y, int l) {
  ellipse(x, y, l, l);
}
```

Uma função pode não precisar produzir (retornar) um valor de saída após ser executada. Nesse caso, usaremos a palavra reservada ***void*** na especificação da função

Em relação a este detalhe, vocês poderão ver em alguns livros ou outros materiais alguns textos fazendo a distinção entre os termos **Função** e **Procedimento**.

Alguns autores costumam usar o termo **Função** para se referir a uma sub-rotina que retorna um determinado valor.

...e o termo **Procedimento** fica para aquelas sub-rotinas que não retornam valor algum

Assim, o comando *ellipse*() é um procedimento. Ou, ainda, uma função do tipo *void*.

A tendência atual é usarmos indistintamente o nome **função** para todas as sub-rotinas, independente de retornarem valor ou não

Uma função não necessariamente precisa receber valores como entrada para ser executada. Nesse caso, deixamos a lista de valores de entrada vazia.

Exemplo: void setup( ) {…}

Relembrando: tanto os valores de entrada como de saída podem ser: ***int, float, boolean,*** ...

Os valores de entrada para as funções recebem o nome de **Parâmetros**, ou ainda, **Argumentos**.

parâmetros

```
void circulo(int x, int y, int l) {
  ellipse(x, y, l, l);
}
```

No Processing, quando vamos trabalhar com funções, devemos notar que todo o código deverá estar dentro de alguma função!

O Processing não aceitará comandos fora de uma função, com exceção do comando de declaração e inicialização de variáveis

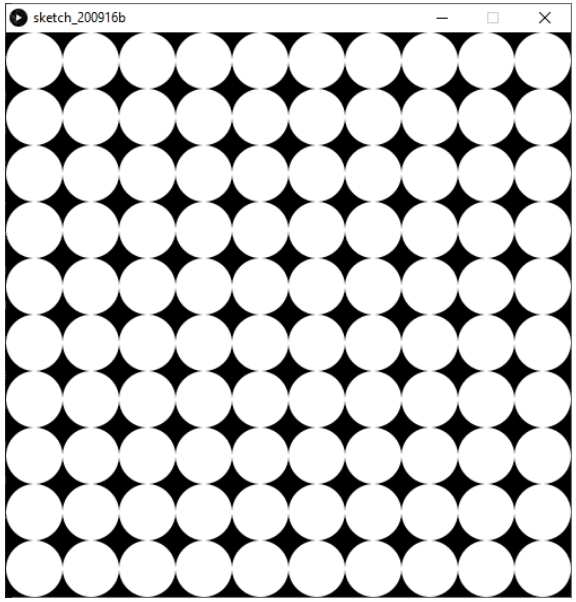
```
int multiplicacao (int a, int b) {
  float resultado;
  resultado = a * b;
  return resultado;
}
Println("Olá Mundo!"); //Não pode!
```

Vamos exercitar um pouco!

Exercício, vamos criar uma função ***desenho*** para o código abaixo

```
void setup() {
  size(500, 500);
  noStroke();
  background(255);
  rectMode(CENTER);
  for (int j=0; j<10; j++)
    for (int i=0; i<10; i++)
      desenho(i*50+25, j*50+25, 50);
}
```

Vamos começar com algo simples como desenhar apenas um círculo na posição (x,y) com largura *l*

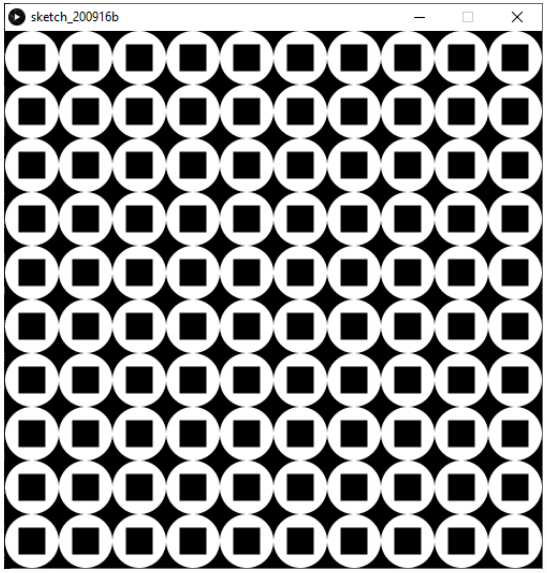


Bem simples, não?

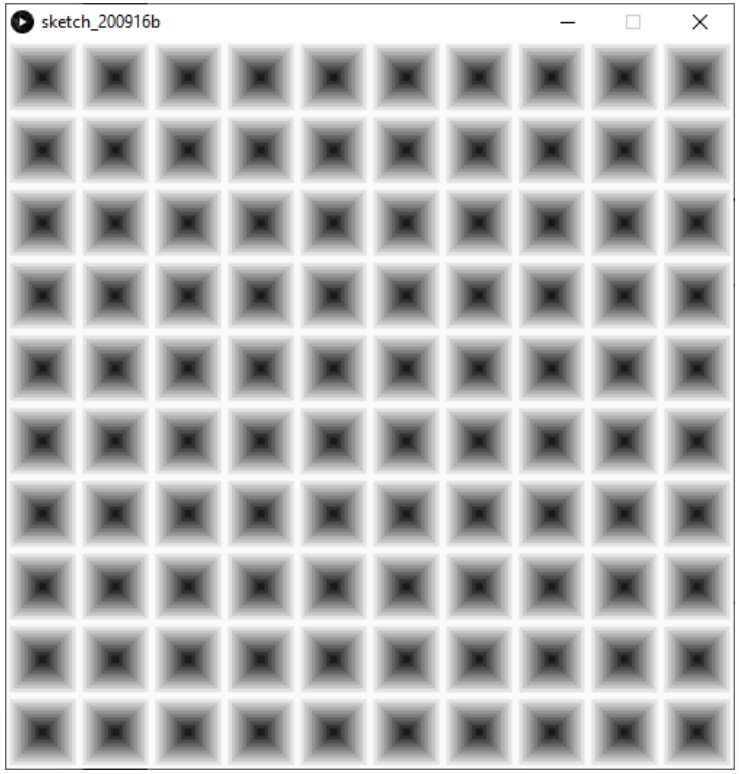
```
void desenho(int x, int y, int l) {
  ellipse(x, y, l, l);
}
```

Vamos elaborar nosso padrão desenhando um quadrado com metade da largura dentro do círculo

```
void desenho(int x, int y, int l) {
  fill(255);
  ellipse(x, y, l, l);
  fill(0);
  rect(x, y, l/2, l/2);
}
```



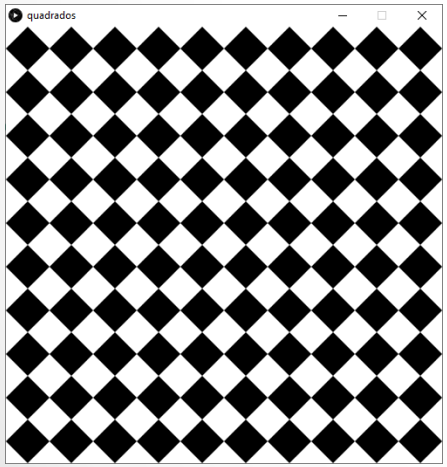
Vamos partir para algo mais elaborado como na imagem abaixo. Percebemos vários quadros com tonalidades de cinza diferentes. Logo precisamos de uma estrutura de repetição na função

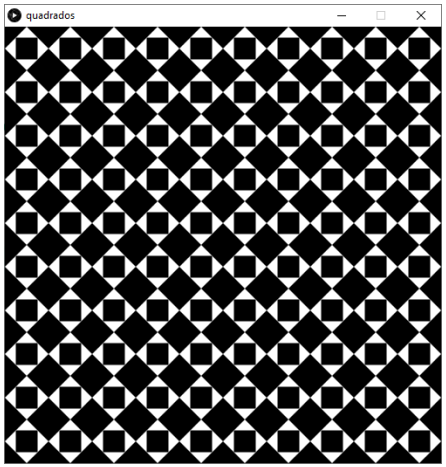


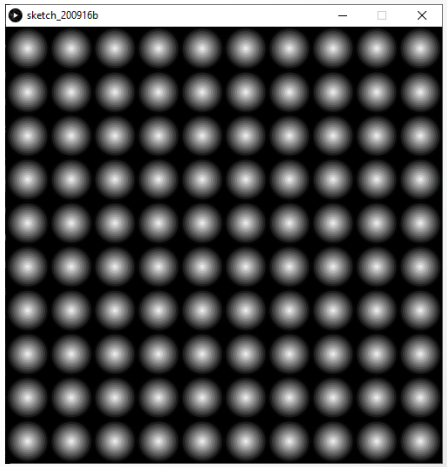
```
void setup() {
  size(500, 500);
  noStroke();
  background(255);
  rectMode(CENTER);
  for (int j=0; j<10; j++)
    for (int i=0; i<10; i++)
      desenho1(i*50+25, j*50+25, 50);
}
void desenho1(int x, int y, int l) {
  for (int i=10; i>0; i--) {
    fill(i*25);
    rect(x, y, i*5, i*5);
  }
}
```

Percebem que podemos criar diferentes padrões apenas modificando a função de desenho

Exercícios, crie uma função de desenho para cada uma das seguintes peças



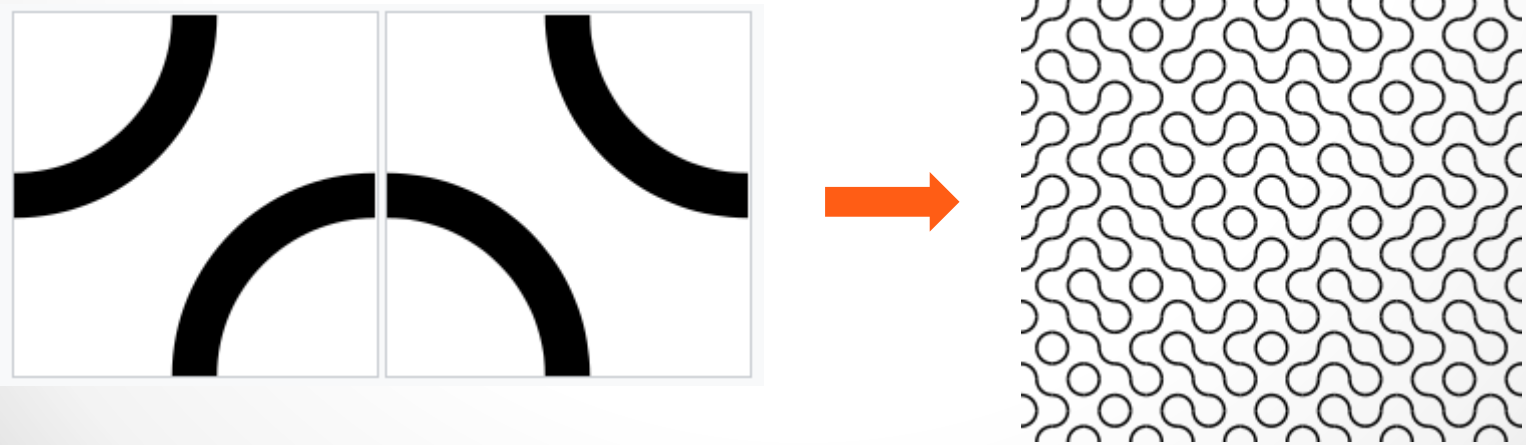




Podemos criar diferentes funções de desenho e alternar a chamada dessas funções

Lembra os padrões de Truchet? Fazer funções para cada um dos elementos abaixo é relativamente fácil.

Imagina agora trabalhar esses elementos e criar algo mais complexo e elaborado, exemplo:



Programação I

George Gomes