Os conceitos de sequência e repetição no *Processing*

A produção em série e a repetição são as marcas registradas do período industrial mecânico. Elas têm a potencialidade de se reproduzirem infinitamente, se, assim o desejarmos. Do mesmo modo, a capacidade de repetição é uma das principais características dos sistemas computacionais.

Uma característica importante da programação é a função de repetição de uma ação. Os comandos formulados pela sintaxe for e function permitem repetir tarefas que, ao serem associadas ao comando if, definem as funções básicas dos sistemas computacionais. Podemos dizer que as decisões dos sistemas computacionais são formuladas por esses comandos que possibilitam estabelecer os caminhos que os algoritmos tomarão para resolver os problemas.

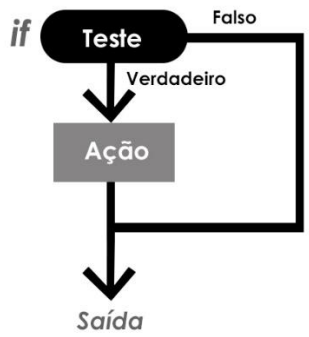
As funções de repetição e o comando de definição dos caminhos definem a lógica a ser utilizada em um programa computacional, o qual, como vimos, está baseado no sistema binário 0 e 1, em que o pulso elétrico é o 1, quando passa energia; se em 0, não passa energia. Essa é toda a estrutura lógica dos computadores.

O comando condicional if, else e elseif

As condições nos sistemas computacionais permitem que um programa faça decisões sobre quais linhas de código serão executadas e quais não o serão. Elas possibilitam que as ações ocorram somente quando uma condição específica é atendida. A função condicional if viabiliza que o programa tome caminhos diferentes dependendo dos valores de suas variáveis. O comando if decide a direção que o programa tomará depois de constatar a validade ou não do teste, isto é, dada uma expressão, o teste verifica se ela é verdadeira ou falsa. Quando a expressão de teste é verdadeira, o código dentro da “chave” é executado. Se a expressão é falsa, o código é ignorado.

No *Processing*, a generalização da função **if** é:

(Entra Figura 46)

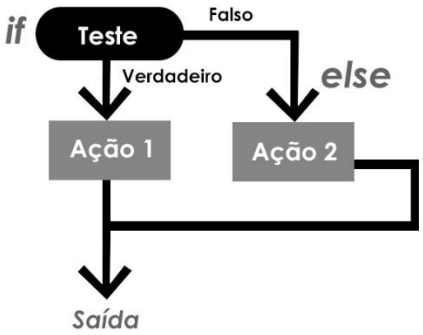



```
If (x < 150) {
        ellipse ( 50, 50, 30, 100);
}
```

Esquema genérico do comando **if** no *Processing* e exemplo

No *Processing*, a generalização da função **if** e **else** é:

(Entra Figura 47)

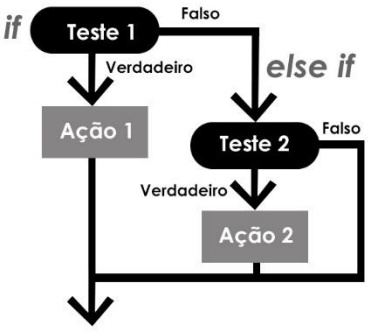



```
If (x < 150) {
        ellipse ( 50, 50, 30, 100);
}
else {
   ellipse ( 50, 50, 50, 200);
}
```

Esquema genérico do comando **if** e **else** no *Processing* e exemplo

Por fim, no *Processing*, a generalização da função **if** e **else if** é:

(Entra Figura 48)




```
If (x < 150) {
   ellipse ( 50, 50, 30, 100);
}
else if (x>150) {
        ellipse ( 50, 50, 50, 200);
}
```

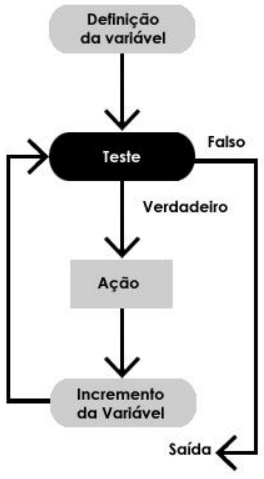
Esquema genérico do comando **if** e **else if** no *Processing* e exemplo

No tópico a seguir, a linguagem de programação do *Processing* vai se concentrar na explicação dos comandos que visam controlar os caminhos a serem percorridos, o fluxo dos programas e suas estruturas iterativas. Os primeiros computadores calculavam de forma mecânica com muita velocidade e precisão na realização dos cálculos repetitivos. Hoje, verificamos que os computadores são interfaces que executam tarefas repetitivas com mais precisão e rapidez. Com base no trabalho dos lógicos Leibniz e Boole, os computadores atuais usam operações lógicas como **e, (*and*)**, **ou (*or*)** e **não (*not*)** para definirem quais linhas de código serão executadas e quais não o serão. Temos também os comandos **function, for** e **void** que definem a execução dos comandos de forma repetitiva.

O comando condicional for

Comecemos pelo comando **for**. As estruturas iterativas são usadas para compactar as linhas de códigos de um programa. O fluxo do código **for**, como mostra o diagrama da Figura 49, detalha a importância central da instrução de teste ao decidir se deve executar o código de ação ou sair da rotina. O diagrama é o formato genérico.

(Entra Figura 49)




```
createCanvas (200,200);
line (20, 20, 20, 180);
line (40, 20, 40, 180);
line (60, 20, 60, 180);
line (80, 20, 80, 180);
line (100, 20, 100, 180);
line (120, 20, 120, 180);
line (140, 20, 140, 180);
```

É equivalente a:

```
createCanvas (200,200);

for (let i = 20; i <150; i = i+20) {
    line (i, 20, i, 180);
}
```

Figura 49 – Esquema genérico do comando **for** no *Processing* e exemplo

Ao lado do diagrama, temos um caso específico que mostra o código extenso e o simplificado com o comando **for**. Os parênteses associados à estrutura incluem três instruções: variável, teste e atualização da variável. As instruções dentro do bloco (ação)

são executadas continuamente, enquanto o teste é avaliado como verdadeiro. A parte variável recebe um valor inicial, no caso do exemplo é o valor 20. Após cada iteração, o programa acrescenta 20 à variável e, assim, o programa é executado na seguinte sequência:

1. A instrução variável é executada (atribui valor 20 para i);
2. O teste é avaliado como verdadeiro ou falso;
3. Se o teste for verdadeiro, o programa continua executando a ação. Se o teste for falso, a rotina ação é abandonada; e
4. O programa abandona a rotina do **"for"** e continua executando o programa.

O comando condicional function setup e function draw

Todos os programas que apresentamos até o momento são executados apenas uma vez. A partir de agora, vamos tratar de programas que necessitam funcionar continuamente, logo são programas que precisam ser controlados na velocidade de execução. Todos os programas que rodam animações ou que respondem às informações ao vivo devem ser processados continuamente. Programas em execução contínua podem usar o *mouse* e o teclado para a entrada de dados.

Assim, todos os programas que são executados continuamente devem ter em sua rotina a função **function draw()**. O código dentro de um bloco **function draw()** é executado em um **loop** (continuamente) até que o botão de parada seja pressionado ou a janela seja fechada.

Um programa pode ter apenas um **function draw()**. Cada vez que a função **function draw()** é executada, o resultado é desenhado na tela e um novo quadro de exibição é mostrado. Em seguida, a rotina **function draw()** começa a executar o bloco novamente a partir da linha inicial do **function draw()** .

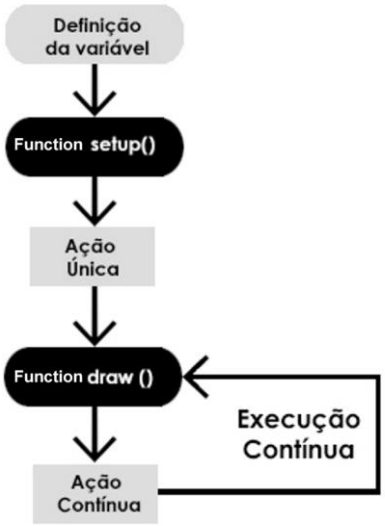
Por padrão, os quadros são desenhados na tela a 60 quadros por segundo (fps). A função **frameRate()** altera e controla o número de quadros exibidos por segundo. A função **frameRate()** controla a velocidade mínima – não é possível acelerar um programa que é executado mais lentamente em função das limitações do equipamento. A variável **frameCount** sempre contém o número de quadros exibidos, assim que o programa começa.

Cada programa pode ter apenas um código de configuração **function setup()** e um **function draw()**. Quando o programa passa pelo código do **function setup()**, o **function**

draw() é executado. O código dentro do bloco **function setup ()** é executado apenas uma única vez. Depois disso, o código do **function draw ()** é processado continuamente até que o programa seja interrompido.

A variável é declarada fora do **function setup ()** e **function draw ()**. Assim, ela será uma variável global que pode ser alterada em qualquer parte do programa. Algumas funções precisam ser executadas uma única vez, portanto os comandos **createCanvas ()** ou **loadImage()** devem ser processados no **function setup()**. As únicas declarações que devem ocorrer fora do **setup ()** e **draw ()** são declarações e atribuições de variáveis. Se um programa desenhar apenas um quadro, ele poderá ser gravado inteiramente dentro do **setup ()**. A única diferença entre **setup ()** e **draw ()** é que o primeiro é executado uma única vez; antes, **draw ()** inicia uma execução contínua (*looping*), portanto as formas desenhadas no **setup ()** aparecerão na tela de *display*. Veja o esquema da Figura 50 que trata das funções **setup** e **draw** de modo genérico:

(Entra Figura 50)




```
let y = 0;

function setup ( ) {
        createCanvas (300,300);
}

function draw ( ) {
      line (0, y, 300, y);
      y = y + 4;
}
```

Esquema genérico do comando **function setup** e **function draw** no *Processing* e exemplo