CAPÍTULO 05

Programando com o Processing

1. Primeiros conceitos

Para escrever um programa em linguagem **Processing** utilizamos apenas os poucos caracteres para construção do código que, após o processo de compilação, produz aplicativos que são controladores de processos industriais até sofisticados sistemas multimídia. Da combinação de letras surgem as palavras reservadas, identificadores, funções de biblioteca, etc.; os caracteres numéricos fornecem a necessária representação de quantidades, tanto em um contexto interno (formatação, parâmetros de inicialização, etc), quanto externo (entrada e saída de dados numéricos) e quanto aos símbolos (* { } / % ^ $ ( ) [ ] ; #...) eles tem usos variados, seja para organizar o texto do programa para definir ao compilador a prioridade de execução de uma rotina ou determinar o fim de uma linha de comando. Alguns símbolos são utilizados como operadores.

2. Ambiente de Programação do Processing

Para resolver problemas computacionais propomos utilizar o aplicativo **Processing** que foi escolhido porque possui as seguintes características:

a) é um software livre;

b) ambiente integrado (editor e compilador) amigável e flexível;

c) extensa biblioteca de funções;

d) permite desenvolver aplicativos em Java e Java Script;

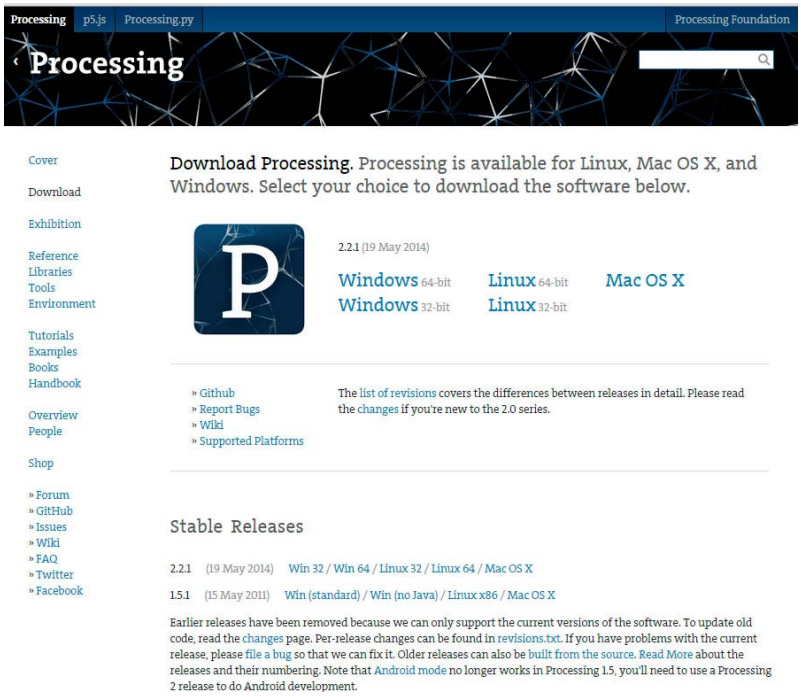
e) permite desenvolver aplicativos em formato de texto;

f) é adequado à iniciante que desconhecem programação;

O ambiente permite utilizar todo o potencial da linguagem, inclusive a exploração de aspectos complexos, porém neste texto abordaremos apenas questões essenciais para o desenvolvimento de aplicativos simples.

Depois de se obter o programa por meio de um *download* do aplicativo no endereço eletrônico https://www.processing.org/. Estão disponíveis as versões

Windows 64-bits, Windows 32-bits, Linux 64-bits, Linux 32-bits e Mac OS X, conforme imagem a seguir.



Depois de instalado, ao executar o ambiente de programação, o **Processing** disponibiliza um editor de texto para a elaboração das linhas de código.

O aplicativo possui duas linhas de menu na primeira temos os comandos File, Edit, Sketch, Tools e Help e, na segunda, temos algumas operações que desecreveremos a seguir.

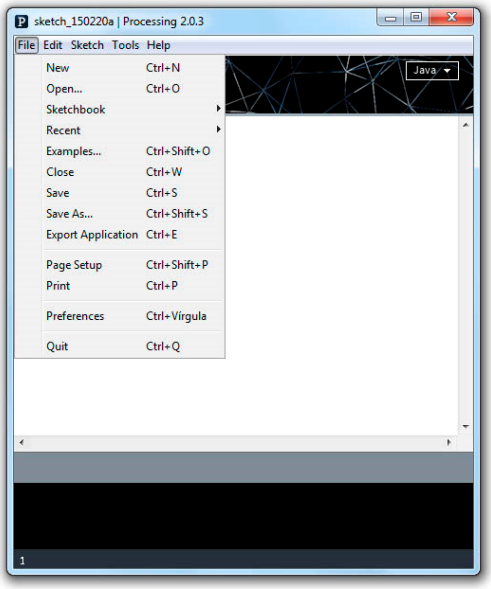


Figura 01 – Funções de Gerenciamento de Arquivos de Programa (File)

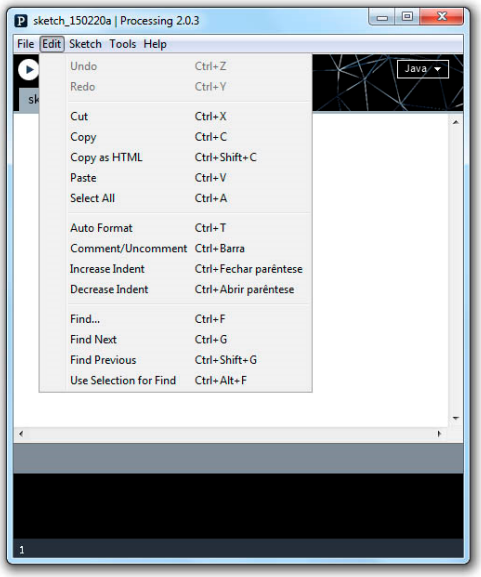


Figura 02 – Funções de Edição (Edit)

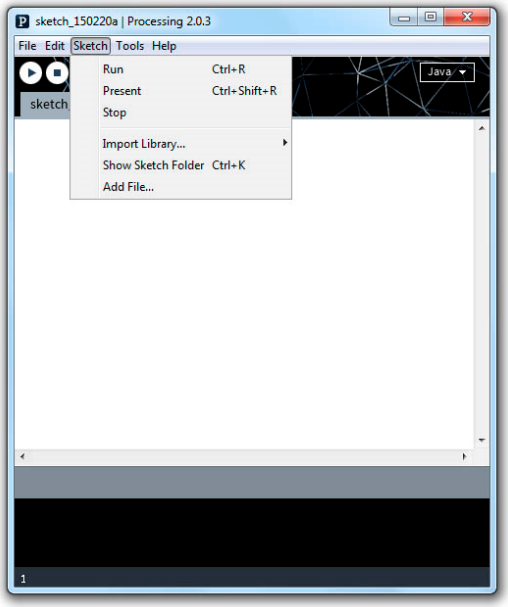


Figura 03 – Funções de Scketch (Edit)

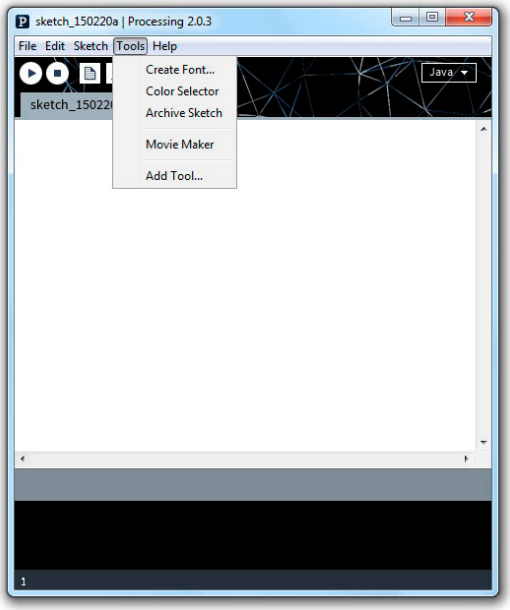


Figura 04 – Funções de Ferramentas (Tools)

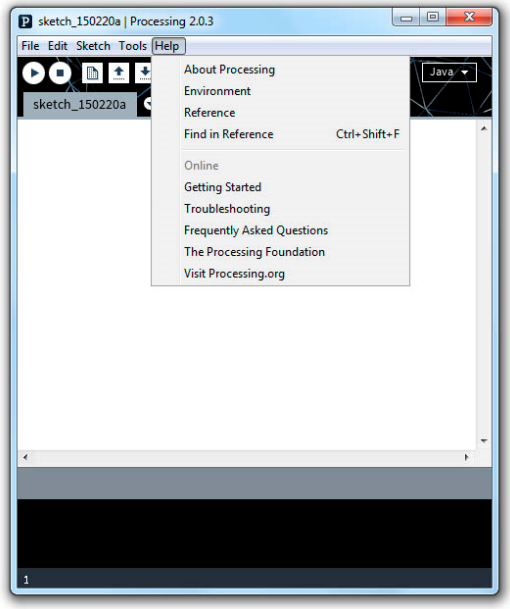


Figura 05 – Funções de Ajuda (Help)

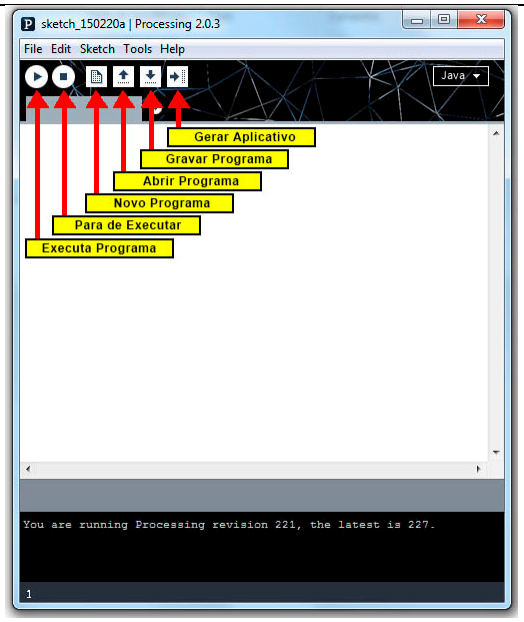


Figura 06 – Segundo Menu

O segundo menu (Fig. 06) permite que se execute um programa elaborado em processing, finaliza a execução de um programa, permite criar um novo programa, abrir um programa realizado, gravar o arquivo de programa elaborado e exportar arquivo gerando um aplicativo.

3. Palavras Chave e Elementos Reservados

As palavras reservadas, em qualquer linguagem, representam tipos, modificadores, especificadores, diretivas e caracterizam a sintaxe da linguagem. Tendo um significado particular dentro da linguagem, as palavras reservadas indicam ao compilador ações específicas que o sistema deverá executar. Como a linguagem **Processing** é sensível à caixa alta ou baixa (maiúscula/minúscula) todos os comandos devem ser escritos em caixa baixa e não podem ser utilizadas com outros propósitos. Todos os comandos da linguagem se resumem a algumas palavras reservadas.

EXPRESSÕES

- Comentários: //, /* */
- Expressões e Afirmações: “;”, “,”
- Comando de Console: print(), println();

COORDENADAS E PRIMITIVAS

- Tamanho das Telas: size();
- Figuras Primitivas: point(), line(), triangle(), quad(), rect(), ellipse();
- Parâmetros de Desenho: background(), fill(), stroke(), noFill(), noStroke();
- Atributos de Desenho: smooth(), noSmooth(), strokeWeight(), strokeCap(), strokeJoin();
- Modos de Desenho: ellipseMode(), rectMode();

VARIÁVEIS

Com as variáveis podemos manipular dados, numéricos ou alfanuméricos, desde a entrada, com sua transformação através do processamento, até a saída dos dados transformados, o que é a essência do que desejamos fazer. Vejamos com mais detalhes esse tipos:

- **boolean** – 1 bit com valor lógico true ou false;
- **byte -** 8 bits -128 to 127;
- **char** - 16 bits 0 to 65535;
- **int -** número inteiro na faixa de -2.147.483.648 a +2.147.483.647 32 bytes;
- **float** - um número racional na faixa de 32 bits 3.40282347E+38 até 3.40282347E+38;
- **true:** verdadeiro;
- **false:** falso;
- **color:** 32 bits 16,777,216 colores.
-

EXPRESSÕES ARITMÉTICAS E FUNÇÕES

+ (soma), **-** (subtração), ***** (multiplicação), **/** (divisão), **%** (módulo);

() (parenteses), **++** (incrementar), **--** (decrementar), **+=** (adicionar e atribuir),

-= (subtrair e atribuir); ***=** (multiplicar e atribuir), **/=** (dividir e atribuir),

- (negação), **round()** (arredondamento), **min()** (mínimo entre números) e **max()** (máximo entre números).

TRANSFORMAÇÕES

Função translate() - A função translate() move a origem da figura do canto superior esquerdo da tela para outro ponto. Ela tem dois parâmetros. O primeiro é a coordenada x e o segundo é a coordenada y. A sintaxe da função translate é translate(x, y). Os valores dos parâmetros x e y são adicionados a quaisquer formas desenhadas após a função ser executada. Se 10 é utilizado como parâmetro para x e 30 é utilizado como parâmetro para y, um ponto desenhado em coordenadas (0,5), será desenhado em coordenadas (10,35).

Função rotate() - A função rotate() gira o sistema de coordenadas de modo que formas podem ser desenhadas na tela em um determinado ângulo. Ele tem um parâmetro que define a quantidade de rotação conforme um ângulo. A função rotação assume que o ângulo é especificado em radianos. As formas são sempre giradas em torno da sua posição em relação à origem (0,0) sendo que o positivo é sentido horário. Tal como acontece com todas as transformações, os efeitos de rotação são acumulativos. Se houver uma rotação de $\pi/4$ radianos e outra de $\pi/4$ radianos, o objeto será desenhado com uma rotação de $\pi/2$ radianos.

4. Uso das Cores – CMYK e RGB

As cores no ***Processing*** são definidas por parâmetros numéricos associados às respectivas sintaxes. Por exemplo: background(), fill() e stroke() são funções específicas. Assim, ao usar as cores com estes parâmetros, eles ficam definidos da seguinte forma: background(valor1, valor2, valor3), fill(valor1, valor2, valor3), fill(valor1, valor2, valor3, alpha), stroke(valor1, valor2, valor3), stroke(valor1, valor2, valor3, alpha), onde os elementos valor1, valor2 e valor3 são parâmetro que variam de 0 a 255 e o valor de alpha varia de 0 a 100% de transparência.

5. Operadores

Quando queremos que uma variável assuma um valor dentro de uma rotina utilizamos a seguinte sintaxe: *nome_da_variável = expressão*, onde a expressão pode ser descrita por um valor constante simples ou utilizar operadores, variáveis e constantes.

Exemplos: **A = 30;**

r1 = (-b+sqrt(b*b-4*a*c))/(2*a);

Os operadores aritméticos da linguagem C++ trabalham de forma semelhante ao que ocorre em outras linguagens e podem ser utilizados em qualquer dado numérico de tipos já mencionados. O que pode surpreender é o tipo do resultado que retorna, dependendo tanto do tipo dos operandos quanto da variável que poderá receber o resultado da expressão, o que nos obriga a ter um cuidado maior pois, ao contrário de outras linguagens, a linguagem C++ não faz verificação de compatibilidade de tipos no momento da compilação e um programa executável pode gerar resultados errôneos que, em muitas ocasiões, somente poderão ser detectados após exaustivos testes. Vejamos os seguintes trechos de programa para ilustrar as situações:

Exemplo 1:

```
{
int a;
a = 2.3+3.5;
cout<<“valor= ”<<a<<endl;
}
```

A primeira impressão é que o compilador irá detectar um erro pois a variável a é de tipo int e os valores são de tipo float, mas o compilador ignora este fato e toma apenas a parte inteira do resultado da expressão (5,8), armazenan- do o valor 5 em a.

Exemplo 2:

```
{
float a; a = 3/2;
cout <<“valor= ”<<a<<endl;
}
```

Podemos ter a impressão de que 3/2 irá produzir o valor 1,5 pois a variável a é de tipo float, mas o valor armazenado em a será 1, quociente inteiro da divisão de 3 por 2.

Observação: se a expressão fosse 3.0/2.0, o resultado seria 1,5.

Relação dos operadores aritméticos

Os operadores estão em ordem de precedência.

+	respectivamente incremento e decremento de uma
-	menos unário
* , / e %	respectivamente multiplicação, divisão e resto de divisão (% trabalha apenas com operandos de tipo
+	respectivamente adição e subtração

Vamos dar uma atenção especial aos operadores incremento e decremento pois geralmente não são encontrados em outras linguagens. O operador incremento (++) soma 1 ao seu operando, de maneira semelhante a "x= x+1", isto não é uma equação, significa que o novo conteúdo de x será seu conteúdo anterior acrescido de uma unidade, supondo o valor de x igual a dois, x receberá o valor 3; utilizando o operador incremento temos x++ e de forma similar podemos ter decremento, ou seja, x=x-1 que pode ser escrito como x--.

O que aconteceria se ao invés de escrevermos x++ escrevêssemos ++x?

É a mesma coisa? Vamos ilustrar a situação:

1º caso:	**2º caso:**
`a = 5;`	`a = 5;`
`b = a++;`	`b = ++a;`

Poderíamos dizer que o valor de b em ambos os casos será 6? Não! Quando o operador de incremento (ou decremento) é utilizado antes do ope- rando (++a),

a operação será executada antes da instrução de atribuição, alte- rando o valor do operando (a) e assim b recebe o valor 6. No primeiro caso, b recebe primeiro o valor 5 (a= 5) e depois o valor de a recebe o incremento. Cuidado então quando utilizar ++ ou --.

Operadores relacionais

Quando desejamos comparar valores numéricos ou não-numéricos, variáveis ou expressões, utilizamos os operadores relacionais, que retornam o valor 1 (true) se a expressão for verdadeira ou 0 (false) se falsa. Utilizamos esses operadores principalmente na definição de estruturas de controle de seleção e controle de repetição, que serão apresentados mais adiante nos capítulos 2 e 3.

Com os operadores relacionais criamos as expressões relacionais que são compostas, obrigatoriamente, por dois operandos de tipos compatíveis e um operador relacional (=, <, >=, etc.) e devolve um valor lógico: falso ou verda- deiro (false ou true), exemplo: x>2, onde "x" é o primeiro operando na forma de uma variável numérica, ">" (maior que) é o operador, que estabelece uma relação entre o primeiro operando e o segundo operando, que é o valor numérico 2 na forma de uma constante.

Caso tenhamos uma expressão relacional matemática na forma: 2<x<7, não podemos escrevê-la desta mesma forma em C++, pois não podemos trabalhar com três operandos. A expressão matemática transforma-se em uma expressão lógica, segundo o seguinte raciocínio: como x deve ser maior que dois e, simultaneamente, menor que sete, temos então a expressão lógica "(2<x) e (x<7)". Portanto, as expressões lógicas são constituídas a partir dos operadores lógicos e expressões relacionais. Observe o exemplo de declaração de uma variável de tipo lógico:

Estes são os operadores relacionais:

Operador	Descrição
==	igualdade
!=	diferença
>	maior que
>=	maior ou igual a
<	menor que

<=	menor ou igual a

Exemplos:

c<5resulta 1 se o conteúdo de afor 4 ou resulta zero se o conteúdo de afor 7.

a!=5resulta 1 se o conteúdo de afor 3 ou resulta zero se o conteúdo de a for 5.

a==5resulta 1 se o conteúdo de afor 5 ou resulta zero se o conteúdo de a for qualquer outro valor.

Operadores lógicos

Existem determinadas situações em que uma expressão relacional simples não atende às necessidade e aí temos que combinar expressões relacionais para obtermos o resultado desejado, construímos então frases lógicas, combinando diversas expressões relacionais. Alguns operadores lógicos**:**

Denominação	**Símbolos**		**Significado**
	Matemático	**C++**	
Negação	**~**	!	não (not)
Conjunção	∧	&&	e
Disjunção	∨	||	ou

Apresentaremos um resumo dos operadores lógicos mais utilizados dentro de um contexto de Lógica Matemática. Neste âmbito, a palavra "proposição" é a expressão de um juízo, que se compõe de:

- um enunciado;
- um valor lógico (falso ou verdadeiro); um
- significado.

Enunciado é o registro da expressão em linguagem escrita, falada ou simbólica (esta última é a forma de expressão da Lógica Matemática). O valor lógico é um atributo da proposição. O significado é a referência a um fato ou ocorrência, é o próprio juízo. Em um sentido mais restrito, podemos considerar uma proposição como uma combinação de expressões relacionais. Uma proposição pode ser simples ou composta.

Negação (~)

A negação de uma proposição *p* é uma proposição cujo valor lógico é *verdadeiro* (V) quando *p* é falsa e *falso* (F) quando *p* é verdadeira, e é representada por "*não p*". De outra forma, "*não p*" tem valor lógico oposto ao de *p*.

Vejamos a tabela-verdade deste operador:

Vejamos a tabela-verdade deste operador:

p	~
V	F
F	V

Exemplos: p: 3 □□3 = 9 (V) e ~p : 3 □□3 □□9
q: 8 > 10 (F) e ~q : 8 □□10 (V)

Na linguagem corrente a negação é feita colocando-se o advérbio "não" antes do verbo ou usando expressões do tipo "*é falso que*" ou "*não é verdade que*" na proposição original. Exemplo:

Seja a proposição p: Ontem choveu. A negação da proposição p (~p) po- derá ser:

1 ~p: Ontem *não* choveu
2 ~p: *É falso que* ontem choveu
3 ~p: *Não é verdade que* ontem choveu.

Vamos a um pequeno exercício mental: como enunciar a negação das seguintes proposições?

p: Todos os alunos são estudiosos q:
Nenhum aluno é estudioso

Conjunção ()

A conjunção de duas proposições simples *p* e *q* é a proposição composta cujo valor lógico é verdadeiro(V) quando as proposições *p* e *q* forem ambas verdadeiras e falso(F) nos demais casos e é representada e lida por "*p e q*", sim- bolicamente "*p ^ q*".

Sua tabela-verdade é a seguinte:

p	q	p ^ q
V	V	V
V	F	F
F	V	F
F	F	F

Exemplos:

2 p: O número □□é maior que 3 (V)
q: O número □□é irracional (V)

p ^ q : O número □□é maior que 3 e o número □□é irracional (V)

3 p: O número □□é menor que 3 (F) q: O número □□é irracional (V)
p ^ q : O número □□é menor que 3 e o número □□é irracional (F)

4 p: O número □□é maior que 3 (V) q: O número □□é racional (F)
p ^ q : O número □□é maior que 3 e O número □□é racional (F)

5 p: O número □□é menor que 3 (F) q: O número □□é racional (F)
p ^ q : O número □□é menor que 3 e O número □□é racional (F)

Disjunção ()

A disjunção de duas proposições simples *p* e *q* é a proposição composta cujo valor lógico é verdadeiro(V) quando pelo menos uma das proposições *p* e *q* for verdadeira e falso(F) quando as proposições *p* e *q* forem ambas falsas e é representada e lida por "*p ou q*", simbolicamente "p v *q*", e sua tabela-verdade é a seguinte:

p	q	p v q
V	V	V
V	F	V
F	V	V
F	F	F

Exemplos:

a) *p* : 2 □□3 □□2 □□3 (*V*)

Tipos de dados não numéricos: char e string

O tipo *char* é utilizado quando desejamos trabalhar com um único caractere existente na tabela ASCII completa (0 ao 255). O tipo *string* foi incorpora- do para facilitar o trabalho do programador quando da necessidade de se manipular palavras, frases e parágrafos.

Vejamos alguns exemplos de declarações:

char cod; string idprod;
cod= 'T';

idprod= "pp12t11"

Sintaxe da Tela de Programação

A partir de agora apresentaremos algumas

- Especificação da coordenada do pixel.

- Formas básicas: ponto, linha, retângulo, elipse.

Este livro vai lhe ensinar como programar no contexto da mídia computacional, e vai utilizar o ambiente de desenvolvimento Processing (http://www.processing.org) como base para todas as discussões e exemplos. Mas antes de tudo isso se torna relevante ou interessante, primeiro temos que canalizar nossos eus da oitava série, retirar um pedaço de papel milimetrado, e desenhar uma linha. ! e menor distância entre dois pontos é uma linha boa moda antiga, e é aí que começamos, com dois pontos sobre que papel milimetrado.

A Figura 1.1 mostra uma linha entre o ponto A (1,0) e do ponto B (4,5). Se você queria dirigir um amigo seu para tirar essa mesma linha, você iria dar-lhes uma mensagem e dizer "desenhar uma linha a partir do ponto de um zero ao ponto 4-5, por favor. "Bem, para o momento, imagine o seu amigo era um computador e você queria instruir este pal digital para exibir essa mesma linha em sua tela. ! e mesmo comando se aplica (só que desta vez você pode ignorar as brincadeiras e você será obrigado a empregar uma formatação precisa). Aqui, a instrução será parecido com este:

line (1,0,4,5);